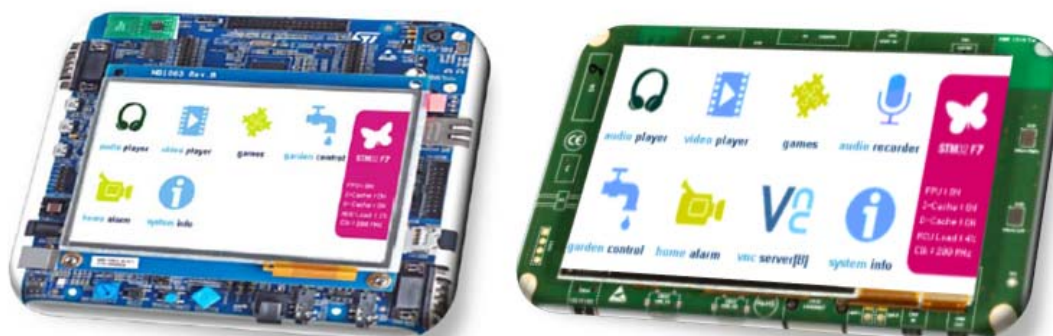


Introduction

The STM32Cube initiative was originated by STMicroelectronics to ease developers' life by reducing the development efforts, time and cost. STM32Cube covers the STM32 portfolio.

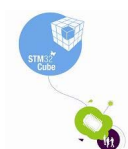
The STM32CubeF7 demonstration platform comes on top of the STM32Cube as a firmware package that offers a full set of software components based on a module architecture allowing re-using them separately in standalone applications. All these modules are managed by the STM32CubeF7 demonstration kernel allowing to dynamically add new modules and access to common resources (storage, graphical components and widgets, memory management, real-time operating system).

The STM32CubeF7 demonstration platform is built around the powerful graphical library STemWin and the FreeRTOS™ real-time operating system and uses almost the whole STM32 capability to offer a large scope of usage based on the STM32Cube HAL BSP and several middleware components.



The architecture was defined with the goal of making from the STM32CubeF7 demonstration core an independent central component which can be used with several RTOS and third party firmware libraries through several abstraction layers inserted between the STM32CubeF7 demonstration core and the several modules and libraries working around.

The STM32CubeF7 demonstration supports STM32F7 Series devices and runs on STM32746G-EVAL, STM32756G-EVAL and STM32746G-Discovery boards. The STM32746G-Discovery demonstration features two modules (audio recorder and VNC server) which are not available on the STM327x6G-EVAL board demonstration.



Contents

1	STM32Cube overview	7
2	Global architecture	8
3	Kernel description	9
3.1	Overview	9
3.2	Kernel initialization	10
3.3	Kernel processes and tasks	10
3.4	Kernel graphical aspect	11
3.5	ST widget add-ons	12
3.5.1	ST animated icon view	12
3.5.2	ST slider skin	13
3.6	Kernel menu management	14
3.7	Module manager	16
3.8	Backup and settings configuration	17
3.9	Storage units	18
3.10	Demonstration repository	21
3.11	Kernel components	22
3.12	Kernel core files	22
3.13	Hardware settings	23
4	Creating a new module	24
4.1	Creating the graphical aspect	24
4.2	Graphics customization	25
4.3	Module implementation	26
4.4	Adding a module to the main desktop	27
5	Demonstration customization and configuration	28
5.1	LCD configuration	28
5.2	Layer management	28
5.3	BSP customization	29
5.3.1	SDRAM configuration	29

	5.3.2	Touch screen configuration	30
6		Performance	32
	6.1	CPU cache	32
	6.2	Multi buffering features	34
	6.3	Multi layers feature	34
	6.4	Hardware acceleration	35
7		Footprint	37
	7.1	STemWin features resources	37
	7.1.1	JPEG decoder	37
	7.1.2	GUI Components	38
8		Demonstration functional description	40
	8.1	Audio player	40
	8.2	Audio recorder	45
	8.3	VNC server	49
	8.4	Video module	54
	8.5	Game	63
	8.6	Garden control	64
	8.7	Home alarm	64
	8.8	System Information	65
9		Revision history	66

List of tables

Table 1.	File system interface functions	18
Table 2.	File system interface APIs	18
Table 3.	Kernel components list	22
Table 4.	Kernel core files list.	22
Table 5.	STM327x6G-EVAL board jumper configuration for demonstration.	23
Table 6.	LCD frame buffer locations	29
Table 7.	Modules footprint	37
Table 8.	RAM requirements for some JPEG resolutions	37
Table 9.	MemoSTemWin components memory requirements	38
Table 10.	Widget memory requirements.	39
Table 11.	Audio module controls	44
Table 12.	Audio module controls	49
Table 13.	Video module controls	54
Table 14.	Video module controls	59
Table 15.	Batch files description	61
Table 16.	Variables description	62
Table 17.	Parameters description	62
Table 18.	Document revision history	66

List of figures

Figure 1.	STM32Cube block diagram	7
Figure 2.	STM32CubeF7 demonstration overview	8
Figure 3.	Kernel components and services	9
Figure 4.	Startup window	11
Figure 5.	Main desktop window	12
Figure 6.	ST animated icon view	13
Figure 7.	Slider skin	13
Figure 8.	Icon view widget	14
Figure 9.	Functionalities and properties of modules	16
Figure 10.	Available storage units	18
Figure 11.	Software architecture	20
Figure 12.	Folder structure	21
Figure 13.	STM32Cube demonstration board	23
Figure 14.	GUI builder overview	24
Figure 15.	Graphics customization	25
Figure 16.	LCDConf location	28
Figure 17.	SDRAM initialization	29
Figure 18.	Touch screen configuration	30
Figure 19.	STM32F7 architecture	32
Figure 20.	STM32F7 performance versus STM32F4	33
Figure 21.	Example of tearing effect	34
Figure 22.	Independent layer management	35
Figure 23.	Audio player module architecture	41
Figure 24.	Audio player process	42
Figure 25.	Audio player module startup	42
Figure 26.	Audio player playlist	43
Figure 27.	Equalizer frame	43
Figure 28.	Hardware connectivity	43
Figure 29.	Audio recorder module architecture	46
Figure 30.	Audio recorder module startup	46
Figure 31.	Start audio recording	47
Figure 32.	Stop audio recording	47
Figure 33.	Play the recorded wave	48
Figure 34.	Hardware connectivity	48
Figure 35.	Video player module architecture	50
Figure 36.	VNC server module startup	51
Figure 37.	Enable/disable secure mode	51
Figure 38.	Start VNC server	51
Figure 39.	Assigned IP address	52
Figure 40.	Entering IP address	52
Figure 41.	Start VNC connection entering the password	52
Figure 42.	Background mode	53
Figure 43.	HW connectivity	53
Figure 44.	Video player module architecture	55
Figure 45.	Video player process	56
Figure 46.	Video player module startup	56
Figure 47.	Video player playlist	57
Figure 48.	Video player playlist popup	57

Figure 49.	Video player frame	58
Figure 50.	Video player control keys	58
Figure 51.	EMF generation environment	60
Figure 52.	JPEG2Movie overview	60
Figure 53.	EMF file generation.	61
Figure 54.	Reversi game module startup	63
Figure 55.	Garden control module startup	64
Figure 56.	Home alarm module startup	64
Figure 57.	Home camera startup	65
Figure 58.	System information startup	65

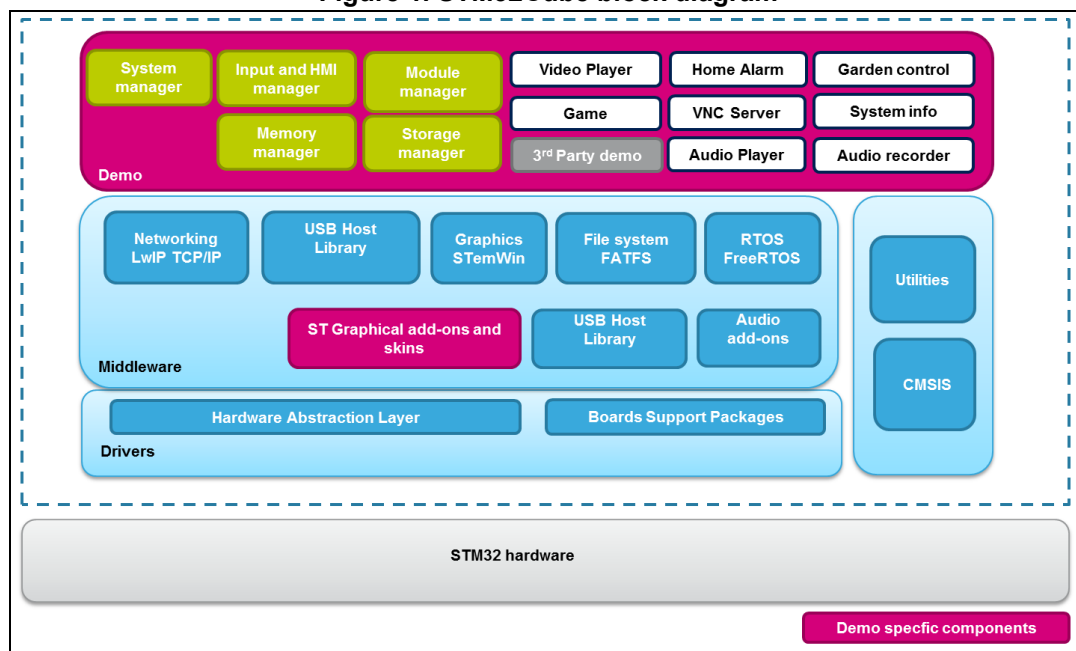
1 STM32Cube overview

The STM32Cube initiative was originated by STMicroelectronics to ease developers' life by reducing development efforts, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows generating C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF7 for STM32F7 Series)
 - The STM32CubeF7 HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
 - A consistent set of middleware components such as RTOS, USB, TCP/IP, graphics
 - All embedded software utilities coming with a full set of examples

Figure 1. STM32Cube block diagram



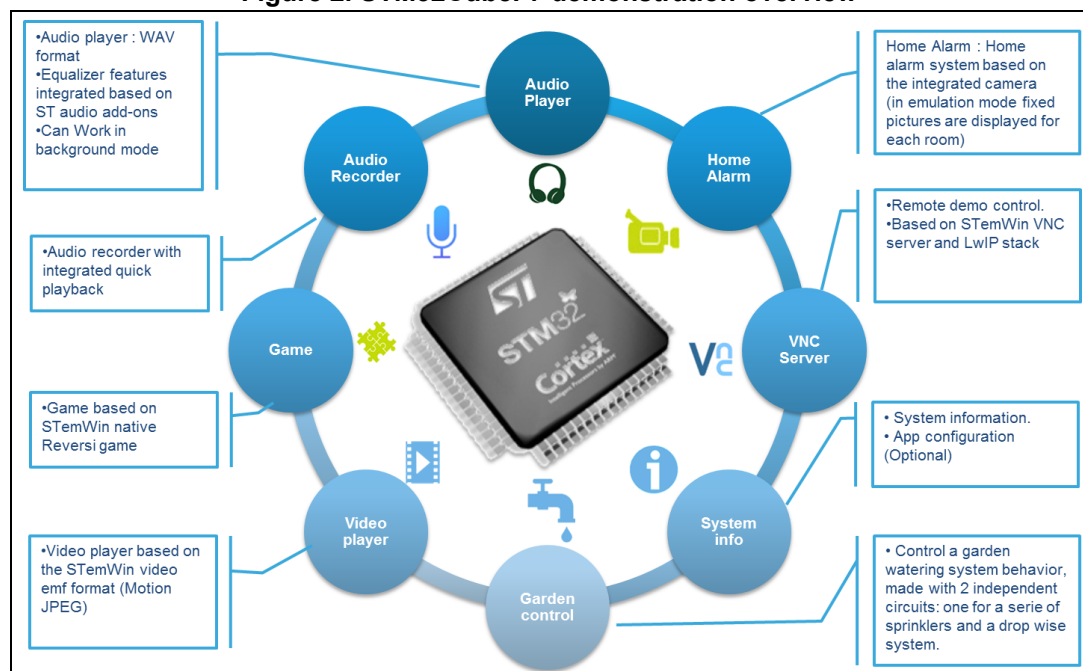
2 Global architecture

The STM32CubeF7 demonstration is composed of a central kernel based on a set of firmware and hardware services offered by the STM32Cube middleware and the several evaluation and discovery boards and a set of modules mounted on the kernel and built in a modular architecture. Each module can be reused separately in a standalone application. The full set of modules is managed by the Kernel which provides access to all common resources and facilitates the addition of new modules as shown in [Figure 2](#) below.

Each module should provide the following functionalities and properties:

1. Icon and graphical aspect characteristics.
2. Method to startup the module.
3. Method to close down safely the module (example: hot unplug for unit storage)
4. Method to manage low-power mode
5. The module application core (main module process)
6. Specific configuration
7. Error management

Figure 2. STM32CubeF7 demonstration overview



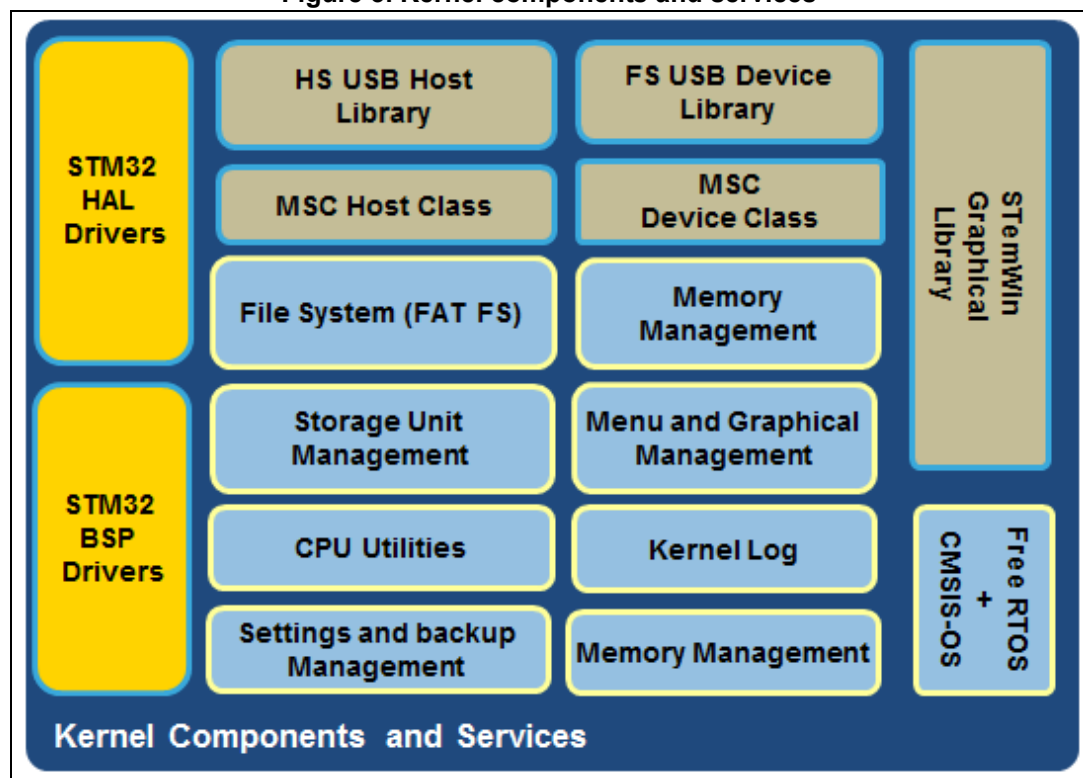
3 Kernel description

3.1 Overview

The role of the demonstration kernel is mainly to provide a generic platform that controls and monitors all the application processes, the kernel provides a set of friendly user APIs and services that allow to the user modules to have access to all the hardware and firmware resources and provide the following tasks and services:

- Hardware and modules initialization:
 - BSP initialization (LEDs, SDRAM, touch screen, CRC, NOR, audio and QSPI)
 - GUI initialization and touch screen calibration
- Memory management
- Graphical resources and main menu management.
- Storage managements (USB Disk Flash memory)
- System monitoring and settings
- CPU utilities (CPU usage, running tasks)

Figure 3. Kernel components and services



3.2 Kernel initialization

The first task of the kernel is to initialize the hardware and firmware resources to make them available to its internal processes and the modules around it. The kernel starts by initializing the HAL, system clocks and then the hardware resources needed during the middleware components:

- LEDs and touchscreen
- SDRAM
- NOR Flash memory
- QSPI memory
- Backup SRAM
- RTC

Note: In the case of STM32746G-Discovery, the external QSPI memory is used to store graphical icons and animated GIF and bitmaps for the overall demonstration otherwise the NOR memory is used.

Once the low level resources are initialized, the kernel performs the STemWin GUI library initialization and prepares the following common services:

- Memory manager
- Storage units
- Module manager
- Kernel Log

Upon full initialization phase, the kernel adds and links the system and user modules to the demonstration core.

3.3 Kernel processes and tasks

The kernel is composed of two main tasks managed by FreeRTOS through the CMSIS-OS wrapping layer:

- GUI thread: this task Initializes the demonstration main menu and then handles the graphical background task when requested by the STemWin:

```
234  /**
235   * @brief Start task
236   * @param argument: pointer that is passed to the thread function as start argument.
237   * @retval None
238   */
239  static void GUIThread(void const * argument)
240  {
241      (...)
242
243      /* Show the main menu */
244      k_InitMenu();
245
246      /* Gui background Task */
247      while(1)
248      {
249          GUI_Exec();
250          osDelay(30);
251      }
252  }
```

- Timer callback: this is the callback of the timer managing periodically the touch screen state, the timer callback is called periodically each 40 milliseconds.

```
262 /**
263  * @brief Timer callback (40 ms)
264  * @param n: Timer index
265  * @retval None
266  */
267 static void TimerCallback(void const *n)
268 {
269     k_TouchUpdate();
270 }
271
```

3.4 Kernel graphical aspect

The STM32Cube demonstration is built around the STemWin Graphical Library, based on SEGGER emWin one. STemWin is a professional graphical stack library, enabling Graphical User Interfaces (GUI) building up with any STM32, any LCD and any LCD controller, taking benefit from STM32 hardware accelerations, whenever possible.

The graphical aspect of the STM32Cube demonstration is divided into two main graphical components:

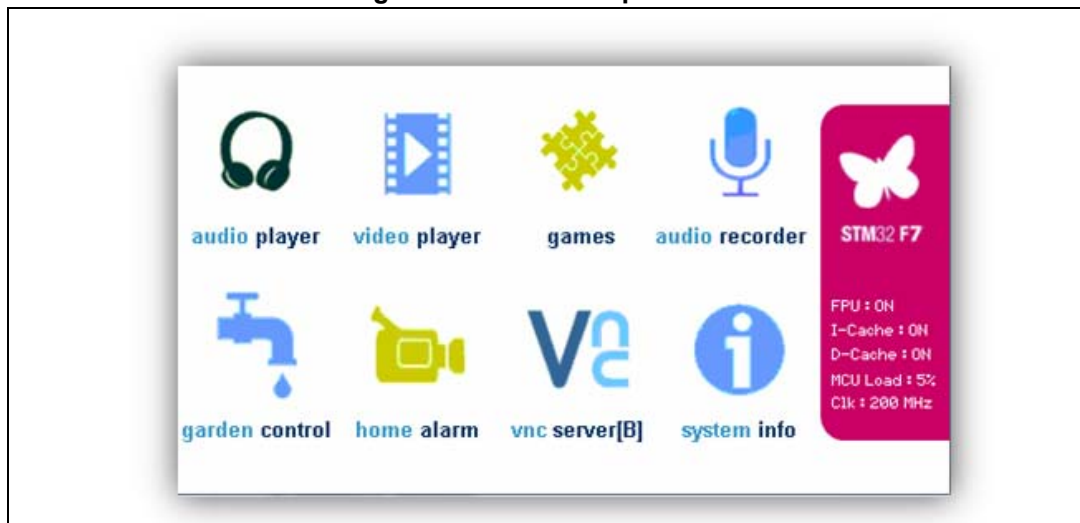
- The startup window ([Figure 4](#)): showing the progress of the hardware and software initialization

Figure 4. Startup window



- The main desktop (shown in [Figure 5](#)), that handles the main demonstration menu and the numerous kernel and modules control.

Figure 5. Main desktop window



Note: The audio recorder and VNC server modules are only applicable for STM32746G-Discovery board.

3.5 ST widget add-ons

The ST_addons binary file provided with STM32F7 demonstration contains new widgets based on STemWin graphical library:

- ST animated icon view
- ST slider skin

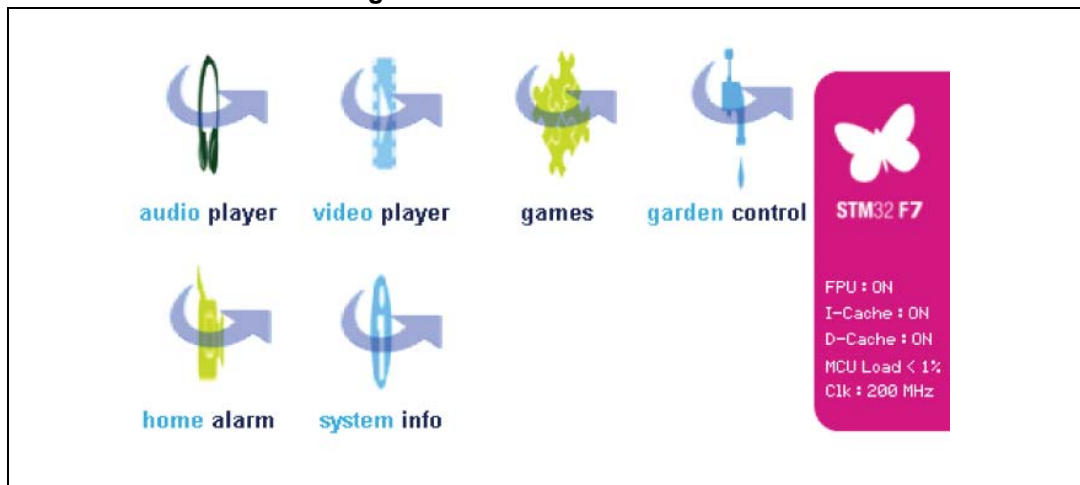
3.5.1 ST animated icon view

A new icon view widget is delivered with STM32F7 demonstration based on STemWin graphical library.

The new widget offers the possibility to turn all the modules icons in the menu after startup with a configured number of frames and configured a delay between each frame.

The new icon view offers also the possibility to configure the module name with two different colors and fonts.

Figure 6. ST animated icon view

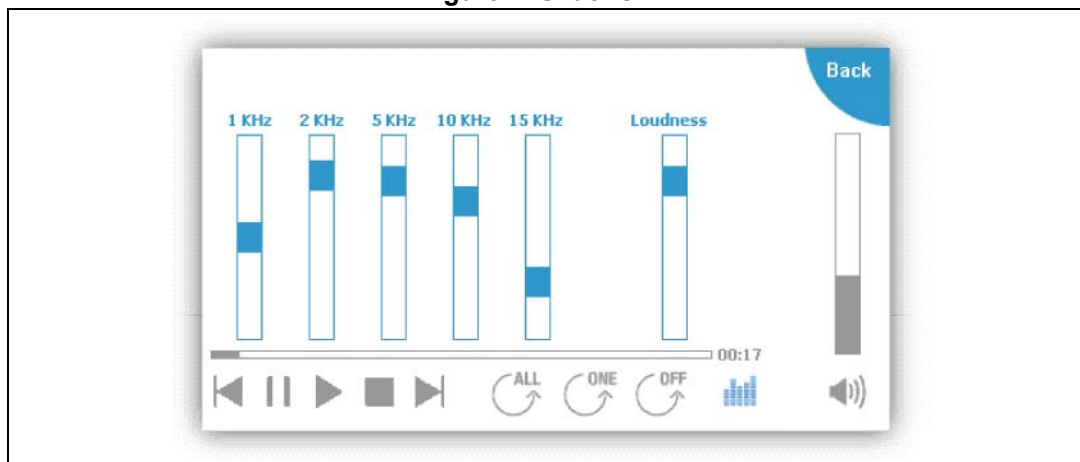


3.5.2 ST slider skin

A new slider skin is delivered with STM32F7 demonstration based on STemWin graphical library.

The new skin offers the possibility to change the slider color and the behavior as shown in the following figure:

Figure 7. Slider skin



3.6 Kernel menu management

The main demonstration menu is initialized and launched by the GUI thread. Before the initialization of the menu the following actions are performed:

- Draw the background image
- Restore general settings from backup memory.
- Setup the main desktop callback to manage main window messages.

The icon view widget: contains the icons associated to added modules. The user can launch a module by a simple click on the module icon.

Figure 8. Icon view widget



A module is launched on simple click on the associated icon by calling to the startup function in the module structure; this is done when a `WM_NOTIFICATION_RELEASED` message arrives to the desktop callback with `ID_ICONVIEW_MENU`:

```

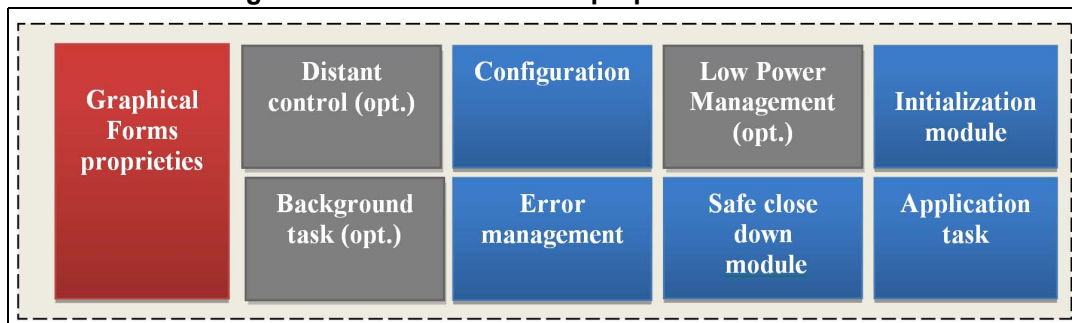
70  /**
71   * @brief Callback routine of desktop window.
72   * @param pMsg: pointer to data structure of type WM_MESSAGE
73   * @retval None
74   */
75  static void _cbBk(WM_MESSAGE * pMsg) {
76
77      (...)
78
79      switch (pMsg->MsgId)
80      {
81      case WM_NOTIFY_PARENT:
82          Id = WM_GetId(pMsg->hWinSrc);
83          NCode = pMsg->Data.v;
84
85          switch (NCode)
86          {
87          case WM_NOTIFICATION_RELEASED:
88              if (Id == ID_ICONVIEW_MENU)
89              {
90                  sel = ST_AnimatedIconView_GetSel(pMsg->hWinSrc);
91
92                  if(sel < k_ModuleGetNumber())
93                  {
94                      ST_AnimatedIconView_SetSel(pMsg->hWinSrc, -1);
95                      if(module_active == 0)
96                      {
97                          module_prop[sel].module->startup(pMsg->hWin, 0, 0);
98                          if(sel != 5)
99                          {
100                              module_active = 1;
101                          }
102                          sel = 0;
103                      }
104                  }
105                  else
106                  {
107                      WM_InvalidateWindow (pMsg->hWinSrc);
108                  }
109              }
110              break;

```

3.7 Module manager

The modules are managed by the kernel which is responsible of initializing the modules, initializing hardware and GUI resources relative to the modules and initializing the common resources such as the storage unit the graphical widgets and the system menu.

Figure 9. Functionalities and properties of modules



Each module should provide the following functionalities and properties:

1. Icon and graphical component structure
2. Method to startup the module
3. Method to close down safely the module (example: Hot unplug for MS flash disk)
4. Method to manage low-power mode (optional)
5. The Application task
6. The module background process (optional)
7. Remote control method (optional)
8. Specific configuration
9. Error management

The modules can be added in run time to the demonstration and use the common kernel resources. The following code shows how to add a module to the demonstration:

```

142      /* Add Modules*/
143      k_ModuleInit();
144
145      /* Link modules */
146      k_ModuleAdd(&audio_player_board);
147      k_ModuleAdd(&video_player_board);
148      k_ModuleAdd(&games_board);
149      k_ModuleAdd(&gardening_control_board);
150      k_ModuleAdd(&home_alarm_board);
151      k_ModuleAdd(&settings_board);

```

A module is a set of function and data structures that are defined in a data structure that provides all the information and pointers to specific methods and functions to the kernel. This later checks the integrity and the validity of the module and inserts its structure into a module table. Each module is identified by a unique ID. When two modules have the same UID, the kernel rejects the second one. The module structure is defined as follows:

```

42 | typedef struct
43 | {
44 |     uint8_t    id;
45 |     const char *name;
46 |     GUI_CONST_STORAGE GUI_BITMAP ** open_icon;
47 |     GUI_CONST_STORAGE GUI_BITMAP ** close_icon;
48 |     void        (*startup) (WM_HWIN , uint16_t, uint16_t );
49 |     void        (*DirectOpen) (char * );
50 | }
51 | K_ModuleItem_Typedef;

```

- Id: unique module identifier.
- Name: pointer to module name
- Open_Icon: pointer to module icon frames (array of bitmap format moving on the right)
- close_Icon: pointer to module icon (array of bitmap format moving on the left), note that the close icon is not yet used in the STM32F7 demonstration.
- Startup: the function that create the module frame and control buttons
- DirectOpen: the function that create the module frame and launch the media associated to the file name selected in the file browser linked to a specific file extension. Note that the direct open functionality is not used in STM32F7 demonstration.

3.8 Backup and settings configuration

The STM32Cube demonstration saves the kernel and modules settings, using the RTC backup register (32 bit data width). With this method the data to be saved should be a 32 bit data and can be defined as a bit field structure as shown in the example:

```

61 | typedef union
62 | {
63 |     uint32_t d32;
64 |     struct
65 |     {
66 |         uint32_t repeat      : 2;
67 |         uint32_t pause      : 2;
68 |         uint32_t mute       : 1;
69 |         uint32_t volume     : 8;
70 |         uint32_t reserved   : 21;
71 |     }b;
72 | }
73 | AudioSettingsTypeDef;

```

The structure can then be handled, by using the two following kernel APIs to save or restore the data from the RTC backup registers:

```

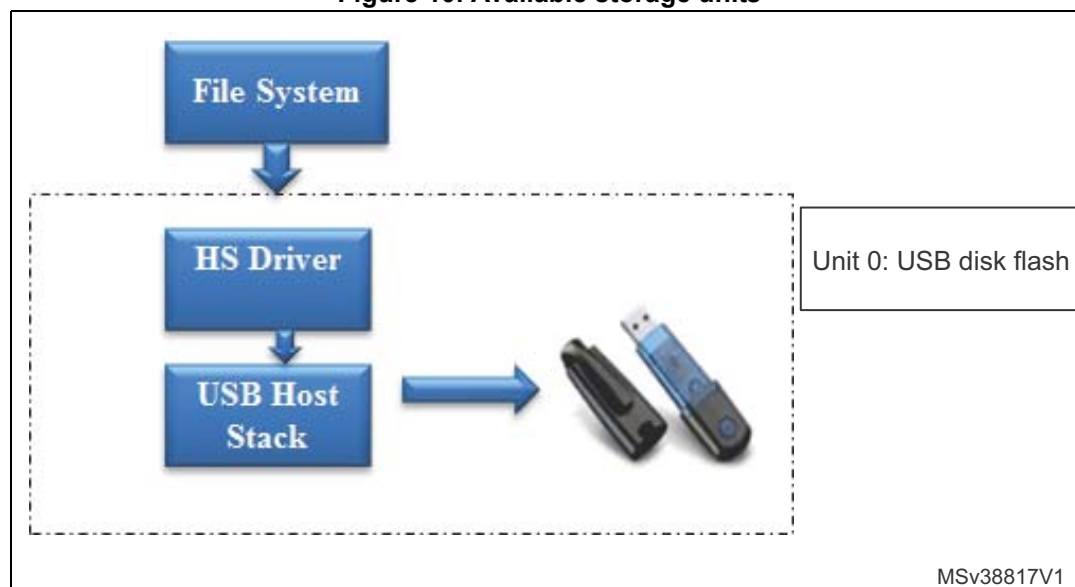
45 | void    k_BkupSaveParameter(uint32_t address, uint32_t data);
46 | uint32_t k_BkupRestoreParameter(uint32_t address);

```

3.9 Storage units

The STM32Cube demonstration kernel offers two storage units that can be used to retrieve audio and video media. The storage unit is initialized during the platform startup and thus they are available to all the modules during the STM32Cube demonstration run time. (Figure 10).

Figure 10. Available storage units



The unit is accessible through the standard I/O operations offered by the FatFS used in the development platform. The USB disk flash unit is identified as the Unit 0 and available only if a USB disk flash is connected on the USB HS connector. The unit is mounted automatically when the physical media are connected to the connector on the board. The implemented functions in the file system interface to deal with the physical storage unit are summarized in Table 1.

Table 1. File system interface functions

Function	Description
disk_initialize	Initialize disk drive
disk_read	Interface function for a logical page read
disk_write	Interface function for a logical page write
disk_status	Interface function for testing if unit is ready
disk_ioctl	Control device dependent features

The full APIs functions set given by the file system interface are listed in Table 2:

Table 2. File system interface APIs

Function	Description
f_mount	Register/unregister a work area
f_open	Open/create a file

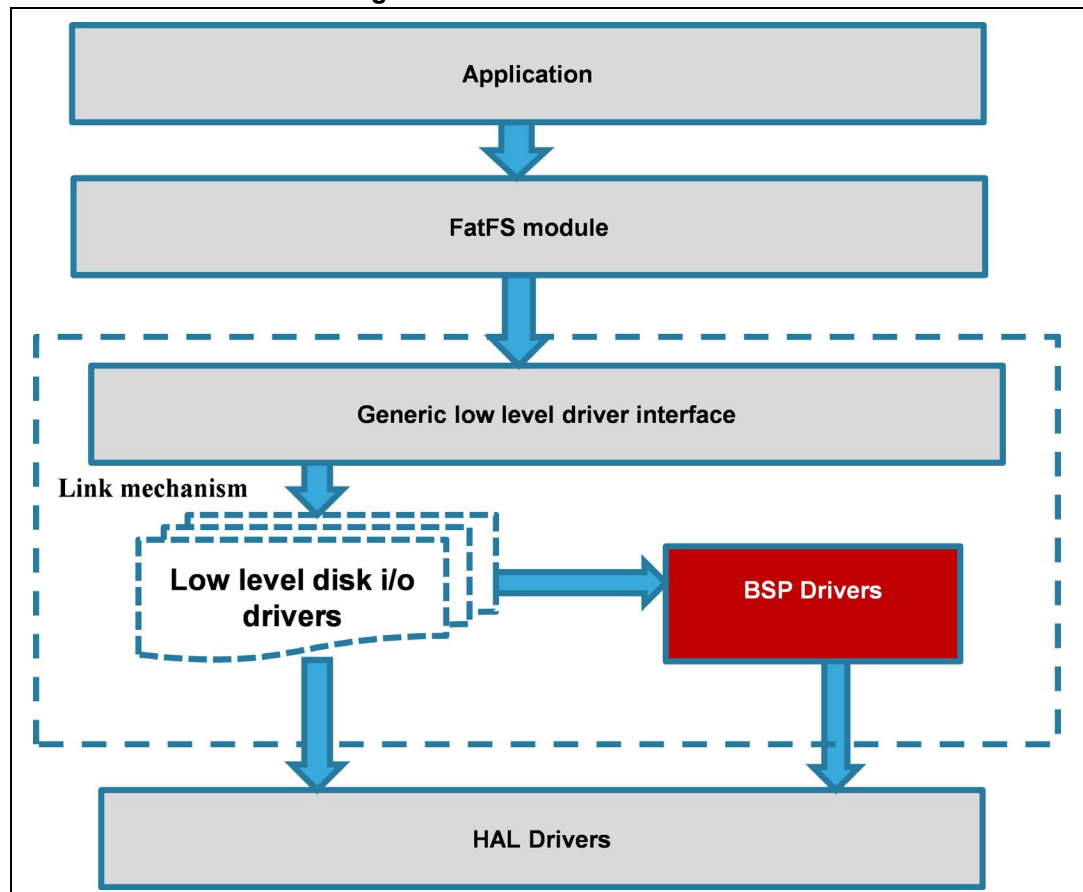
Table 2. File system interface APIs (continued)

Function	Description
f_close	Close a file
f_read	Read file
f_write	Write file
f_lseek	Move read/write pointer, Expand file size
f_truncate	Truncate file size
f_sync	Flush cached data
f_opendir	Open a directory
f_readdir	Read a directory item
f_getfree	Get free clusters
f_stat	Get file status
f_mkdir	Create a directory
f_unlink	Remove a file or directory
f_chmod	Change attribute
f_ftime	Change timestamp
f_rename	Rename/Move a file or directory
f_mkfs	Create a file system on the drive
f_forward	Forward file data to the stream directly
f_chdir	Change current directory
f_chdrive	Change current drive
f_getcwd	Retrieve the current directory
f_gets	Read a string
f_putc	Write a character
f_puts	Write a string
f_printf	Write a formatted string

For the FAT FS file system, the page size is fixed to 512 bytes. USB disk flashes with higher page size are not supported.

The storage unit is built around the USB host library in high speed, the software architecture is shown in [Figure 11](#).

Figure 11. Software architecture

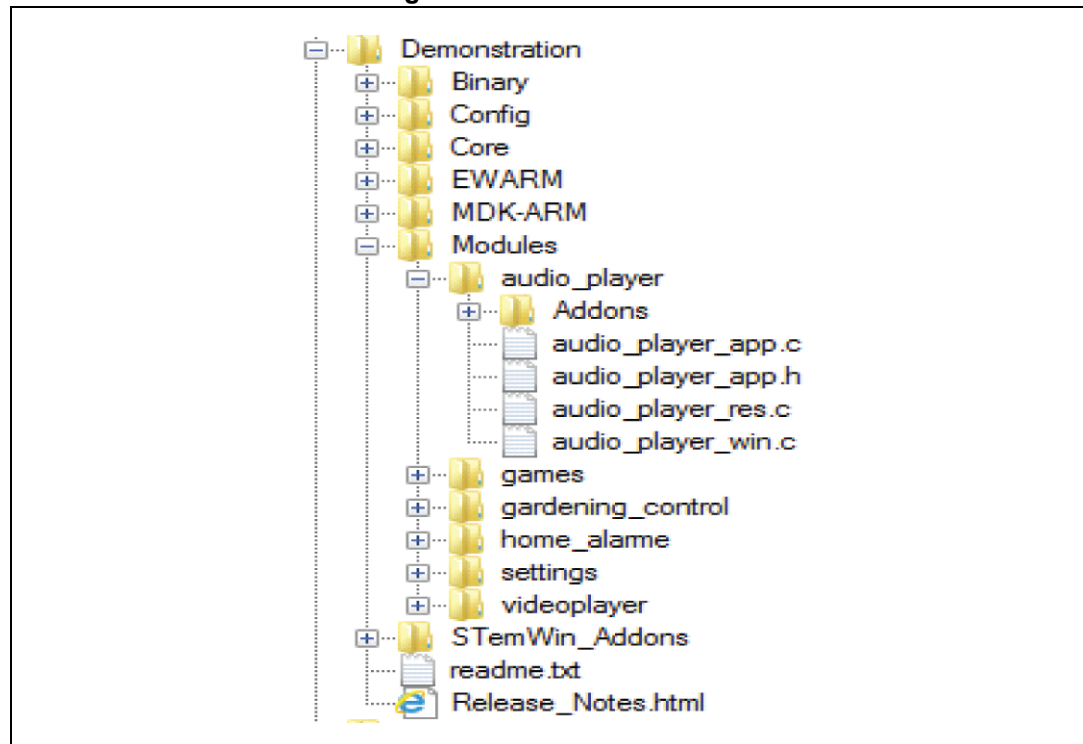


The FatFS is mounted upon the USB host mass storage class to allow an abstract access to the physical media through standard I/O methods.

3.10 Demonstration repository

The STM32Cube is a component in the STM32Cube package. [Figure 12](#) shows the demonstration folder organization:

Figure 12. Folder structure



The demonstration sources are located in the projects folder of the STM32Cube package for each supported board. The sources are divided into six groups described as follows:

- Core: contains the kernel files
- Modules: contains the module core manager and the graphical aspect and the windowing management of the modules.
- Binary: demonstration binary file in Hex format
- Config: all middleware's components and HAL configuration files
- Project settings: a folder per tool chain containing the project settings and the linker files.
- STemWin_Addons: contains the binary file for added widgets based on STemWin graphical library.

3.11 Kernel components

Table 3. Kernel components list

Function	Description
Kernel core	Kernel core and utilities
Modules	User and system modules
STM32 HAL Drivers	STM32Cube HAL driver relative to the STM32 device under use
BSP Drivers	Evaluation board (or discovery kit) BSP drivers
CMSIS	CMSIS CortexM [®] Device Peripheral Access Layer System
FatFS	FATFS File system
FreeRTOS	FreeRTOS Real-Time Operating System
STemWin	STemWin graphical library
USBD_Library	USB device library (Mass Storage Class)
USBH_Library	USB host library (Mass Storage Class)
LWIP	LWIP Library

3.12 Kernel core files

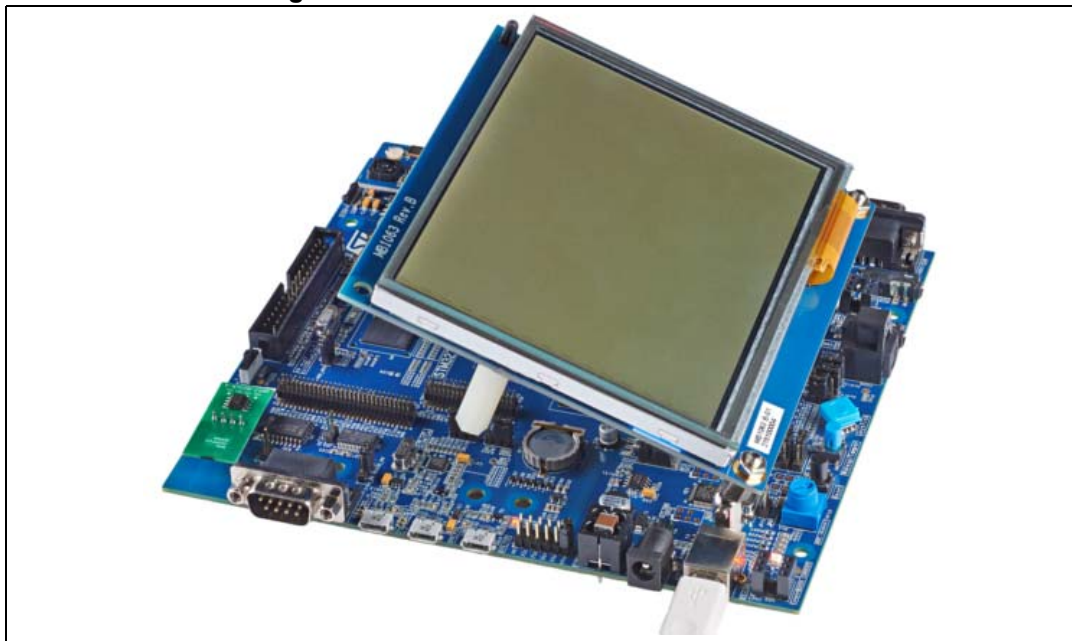
Table 4. Kernel core files list

Function	Description
main.c	Main program file
stm32fxxx_it.c	Interrupt handlers for the application
k_bsp.c	Provides the kernel BSP functions
k_menu.c	Kernel menu and desktop manager
k_module.c	Module manager
k_modules_res.c	Common modules resources
k_rtc.c	RTC and backup manager
k_startup.c	Demonstration startup windowing process
k_storage	Storage units manager
startup_stm32fyyyxx.s	Startup file
cpu_utils.c	CPU load calculation utility

3.13 Hardware settings

- The STM32Cube demonstration supports STM32F7 Series devices and runs on STMicroelectronics STM32756G-EVAL, STM32746G-EVAL and STM32746G-Discovery demonstration boards.
No specific hardware settings or jumper configurations are needed to have the demonstration running on the STM32746G-Discovery board at the exception of the power supply caution (details are provided in the note below).

Figure 13. STM32Cube demonstration board



- The following table summarizes the STM327x6G-EVAL board jumper configurations to have the demonstration running.

Table 5. STM327x6G-EVAL board jumper configuration for demonstration

Board	Jumper	Position description
STM327x6G-EVAL	JP10	Must be not fitted (NOR write protection)
	JP18	<1-2> position (used for audio player module)
	JP19	<1-2> position (used for audio player module)
	JP21	<1-2> position (used for audio player module)
	JP22	<1-2> position (used for audio player module)

Note: The demonstration runs @200 MHz since the SDRAM clock is limited to 100 MHz.
The board must be powered by an external power supply: 5V > 1A (ST-LINK USB not enough to power the board).
In the case of the STM32F746G-Discovery JP1 must be put in 5V ext and the external power supply must be connected to JP2. (for more details please refer to 'Programing/debugging when the power supply is not from ST-LINK (5V link)' section of the UM1907 user manual).

4 Creating a new module

A module is composed of two main parts:

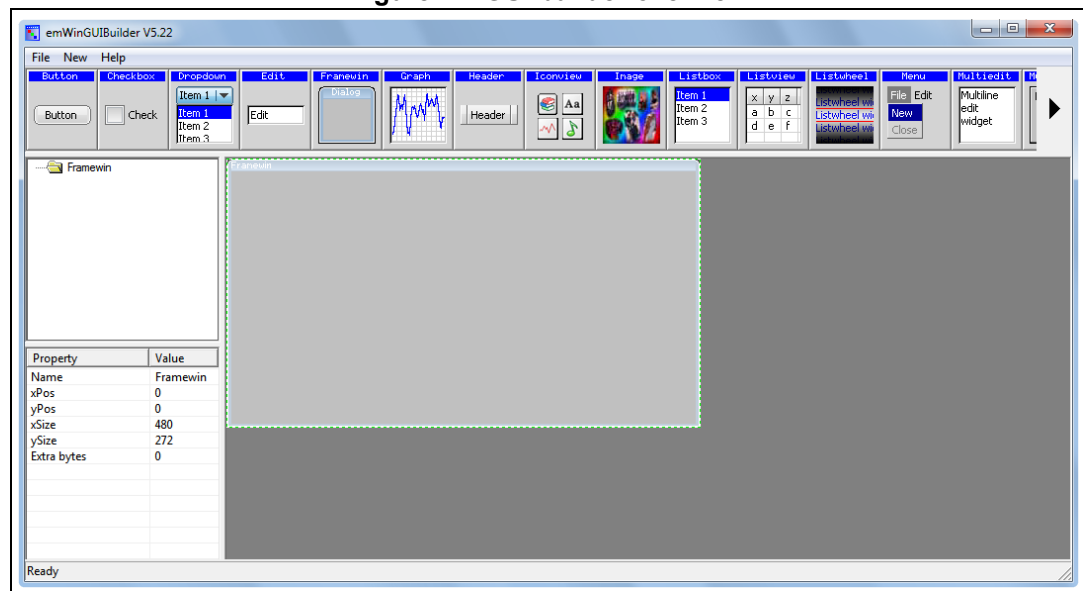
- Graphical aspect: the main window frame and module controls
- Functionalities: module functions and internal processes

4.1 Creating the graphical aspect

The graphical aspect consists of the main frame window in addition to the set of the visual elements and controls (buttons, check boxes, progress bars...) used to control and monitor the module functionalities.

The demonstration package includes the GUI builder, a useful PC application used to easily and quickly create the module frame window and all its components.

Figure 14. GUI builder overview



The GUI builder needs only a few minutes to totally design the module appearances using "drag and drop" commands and then generate the source code file to be included into the application.

The file generated is composed of the following main parts:

- A resource table: it's a table of type GUI_WIDGET_CREATE_INFO, which specifies all the widgets to be included in the dialog and also their respective positions and sizes.
- A dialog callback routine: described more in detail in [Section 3.3](#) (it is referred to as "main module callback routine").

4.2 Graphics customization

After the basic module graphical appearance is created, it is then possible to customize some graphical elements, such as the buttons, by replacing the standard aspect by the user defined image. To do this, a new element drawing callback should be created and used instead of the original one.

Below an example of a custom callback for the play button:

```

363  /**
364   * @brief  callback for play button
365   * @param  pMsg: pointer to data structure of type WM_MESSAGE
366   * @retval None
367   */
368  static void _cbButton_play(WM_MESSAGE * pMsg) {
369      switch (pMsg->MsgId) {
370          case WM_PAINT:
371              _OnPaint_play(pMsg->hWin);
372              break;
373          default:
374              /* The original callback */
375              BUTTON_Callback(pMsg);
376              break;
377      }
378  }

```

On the code portion above, the `_OnPaint_play` routine contains just the new button drawing command.

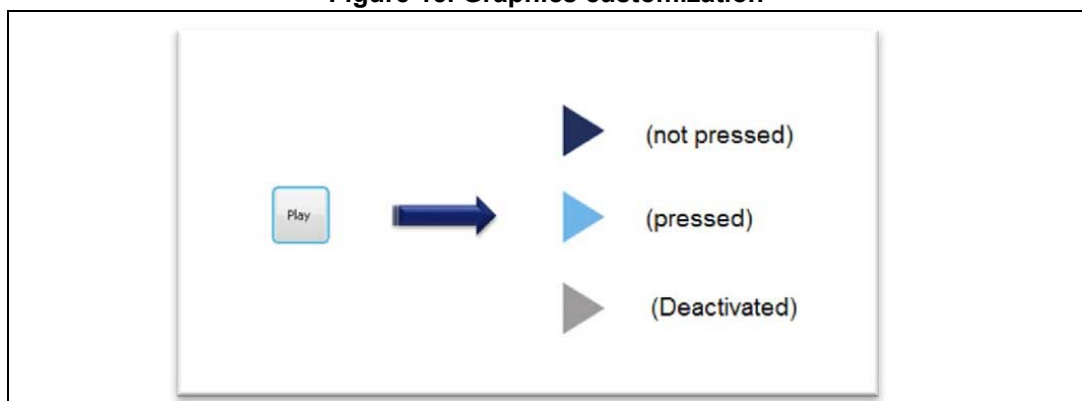
Note that the new callback should be associated to the graphical element at the moment of its creation, as shown below:

```

1579  hItem = BUTTON_CreateEx(112, 420, 40, 40, pMsg->hWin, WM_CF_SHOW, 0, ID_PLAY_BUTTON);
1580  WM_SetCallback(hItem, _cbButton_play);

```

Figure 15. Graphics customization



4.3 Module implementation

Once the graphical part of the module is finalized, the module functionalities and processes can be added. It begins with the creation of the main module structure as defined in [Section 3.7: Module manager](#).

Then, each module has its own startup function which simply consists of the graphical module creation, initialization and link to the main callback:

```

1469  /**
1470   * @brief Module window Startup
1471   * @param hWin: pointer to the parent handle.
1472   * @param xpos: X position
1473   * @param ypos: Y position
1474   * @retval None
1475   */
1476  static void Startup(WM_HWIN hWin, uint16_t xpos, uint16_t ypos)
1477  {
1478      GUI_CreateDialogBox(_aDialogCreate, GUI_COUNTOF(_aDialogCreate), _cbDialog, hWin, xpos, ypos);
1479  }

```

In the example above cbDialog refers to the main module callback routine. Its general skeleton is structured like the following:

```

931  /**
932   * @brief Callback routine of the dialog
933   * @param pMsg: pointer to data structure of type WM_MESSAGE
934   * @retval None
935   */
936  static void _cbDialog(WM_MESSAGE * pMsg) {
937      switch (pMsg->MsgId) {
938          case WM_INIT_DIALOG:
939              /* Initialize graphical elements and restore backup parameters if any */
940          case WM_NOTIFY_PARENT:
941              Id = WM_GetId(pMsg->hWinSrc);
942              NCode = pMsg->Data.v;
943              switch (Id) {
944                  case ID_BUTTON:
945                      switch (NCode) {
946                          case WM_NOTIFICATION_RELEASED:
947                              /* Operation associated to the button */
948                      }
949              }
          (...)

```

The list of windows messages presented above in the code sections (WM_INIT_DIALOG and WM_NOTIFY_PARENT) is not exhaustive, but represents the essential message IDs used:

- "WM_INIT_DIALOG: allows initializing the graphical elements with their respective initial values. It is also possible here to restore the backup parameters (if any) that will be used during the dialog procedure.
- "WM_NOTIFY_PARENT: describes the dialog procedure, for example: define the behavior of each button.

The full list of window messages can be found in the WM.h file.

4.4 Adding a module to the main desktop

Once the module appearance and functionality are defined and created, the module still needs to be added to the main desktop view. This is done by adding the module to the list (structure) of menu items: `module_prop[ ]`, defined into `k_module.h`.

To do this, `k_ModuleAdd()` function should be called just after the module initialization into the `main.c` file.

Note that the maximum modules number in the demonstration package is limited to 15; this value can be changed by updating `MAX_MODULES_NUM` defined into `k_module.c`.

5 Demonstration customization and configuration

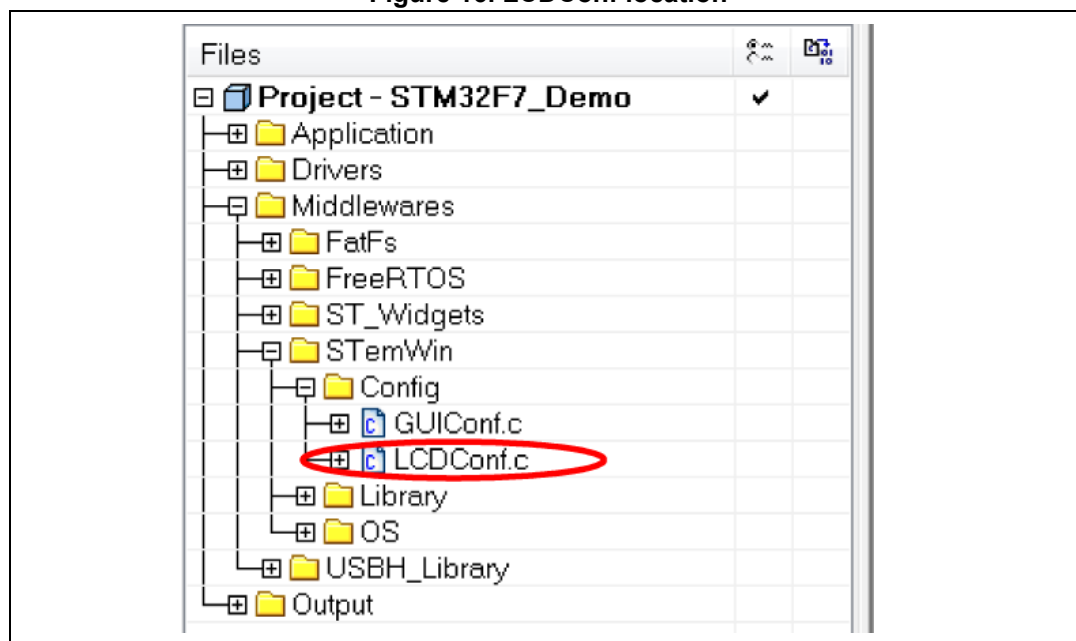
5.1 LCD configuration

The LCD is configured through the LCDConf.c file, see [Figure 16](#). The main configuration items are listed below:

- Multiple layers
 - The number of layers to be used defined using GUI_NUM_LAYERS.
- Multiple buffering
 - If NUM_BUFFERS is set to a value "n" greater than 1, it means that "n" frame buffers will be used for drawing operation (see [Section 6.2](#) for impact of multiple buffering on performance).
- Virtual screens
 - If the display area is greater than the physical size of the LCD, NUM_VSCREENS should be set to a value greater than 1. Note that virtual screens and multi buffers are not allowed together.
- Frame buffers locations:

The physical location of frame buffer is defined through LCD_LAYERX_FRAME_BUFFER.

Figure 16. LCDConf location



5.2 Layer management

In the demonstration package, GUI_NUM_LAYERS is set to two (both layers are used):

- "Layer 0 is used for the main desktop display
- "Layer 1 is used for the video player module playback

Such display separation will help lighten the CPU load during the refresh tasks.

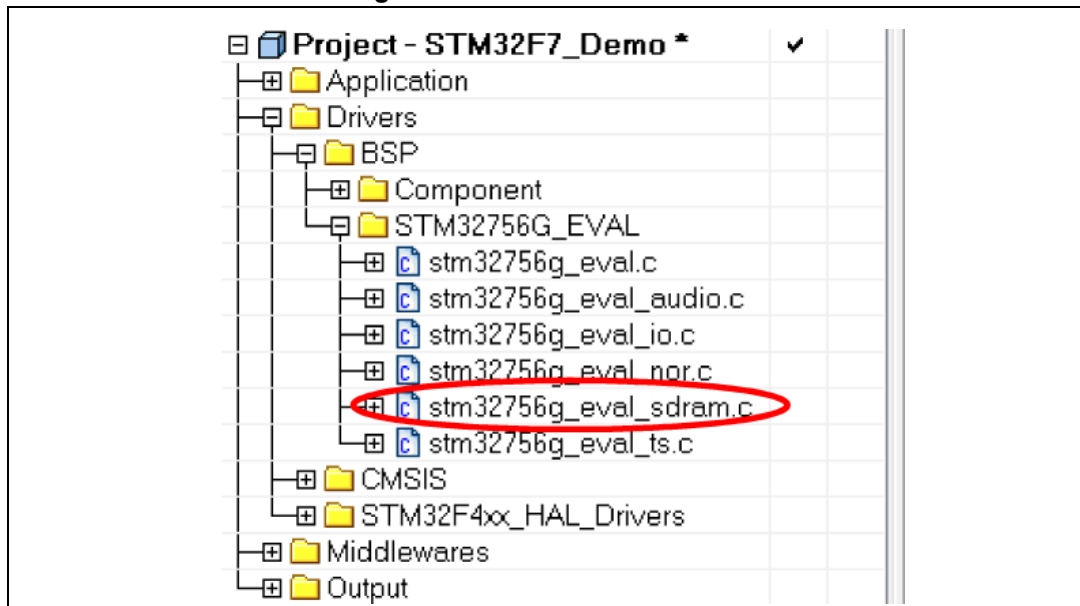
5.3 BSP customization

5.3.1 SDRAM configuration

The SDRAM capacity is 1Mbyte x 32 bit x 4 banks.

The BSP SDRAM driver offer a set of functions to initialize, read/write in polling or DMA mode.

Figure 17. SDRAM initialization



The SDRAM external memory must be initialized before the GUI initialization to allow his use as LCD layers frame buffer.

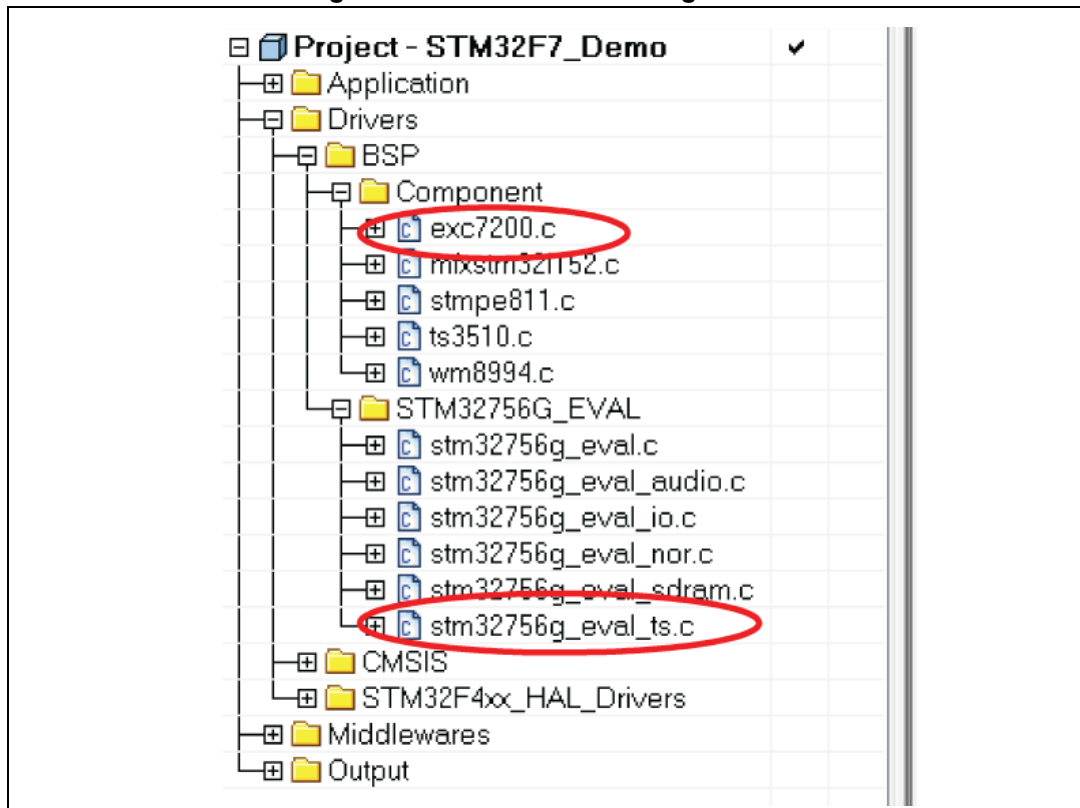
Table 6. LCD frame buffer locations

Layer	Address
LCD Layer0	0xC0000000
LCD Layer1	0xC0400000

5.3.2 Touch screen configuration

The touch screen is controlled by BSP TS driver which uses the exc7200 component in case of STM32746G-EVAL and STM32756G-EVAL boards.

Figure 18. Touch screen configuration



For STM32746G-Discovery, the touch screen is controlled by BSP TS driver which uses the ft5336 component.

The touch screen is initialized in 'k_BspInit' following the used screen resolution as shown in the code below:

```
51  /**
52   * @brief Initializes LEDs, SDRAM, touch screen, CRC and SRAM.
53   * @param None
54   * @retval None
55   */
56  void k_BspInit(void)
57  {
58      (...)
59      /* Initialize the Touch screen */
60      BSP_TS_Init(640, 480);
61      (...)
62  }
63
64  /**
65   * @brief Read the coordinate of the point touched and assign their
66   *        value to the variables u32_TSXCoordinate and u32_TSYCoordinate
67   * @param None
68   * @retval None
69   */
70  void k_TouchUpdate(void)
71  {
72      static GUI_PID_STATE TS_State = {{0}, {0}, {0}, {0}};
73
74      BSP_TS_GetState((TS_StateTypeDef *)&ts);
75      (...)
76      GUI_TOUCH_StoreStateEx(&TS_State);
77      (...)
78  }
```

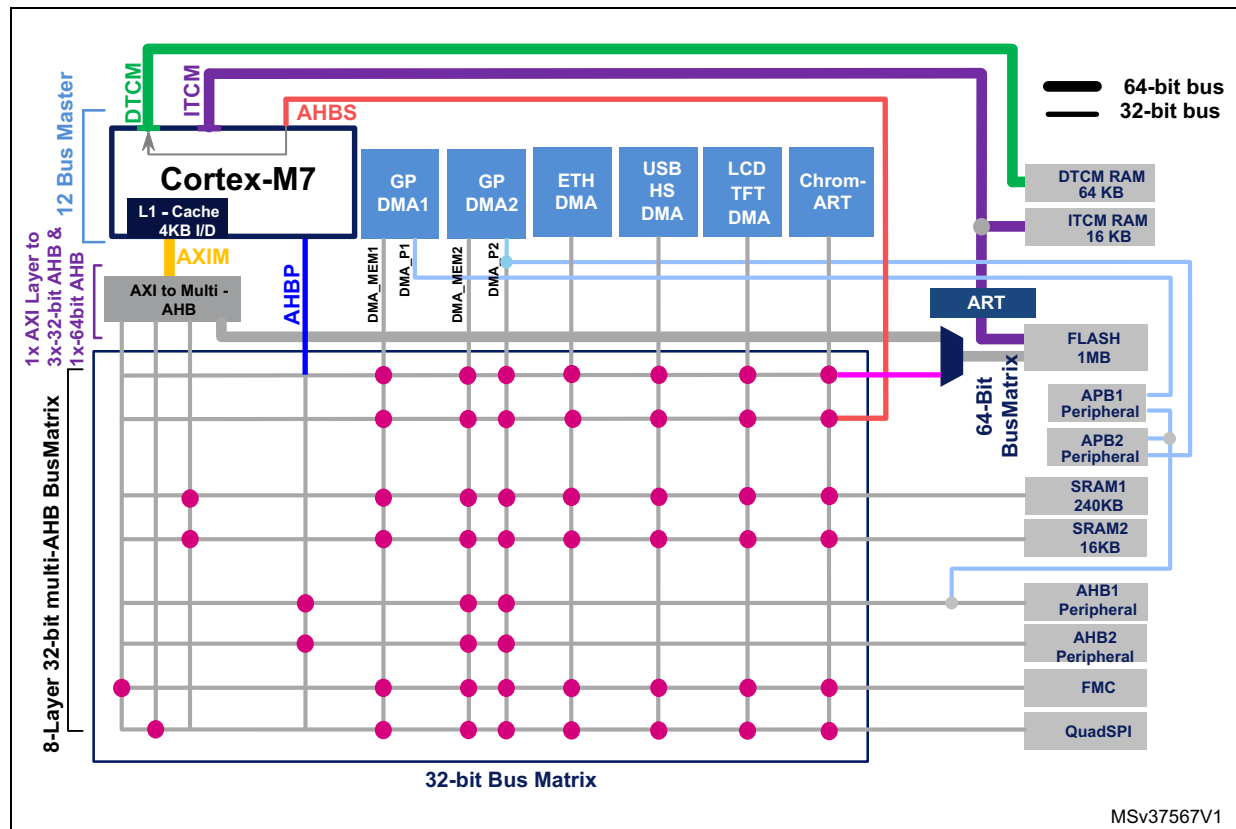
6 Performance

6.1 CPU cache

The STM32F7 demonstration benefits from cortex M7 performance

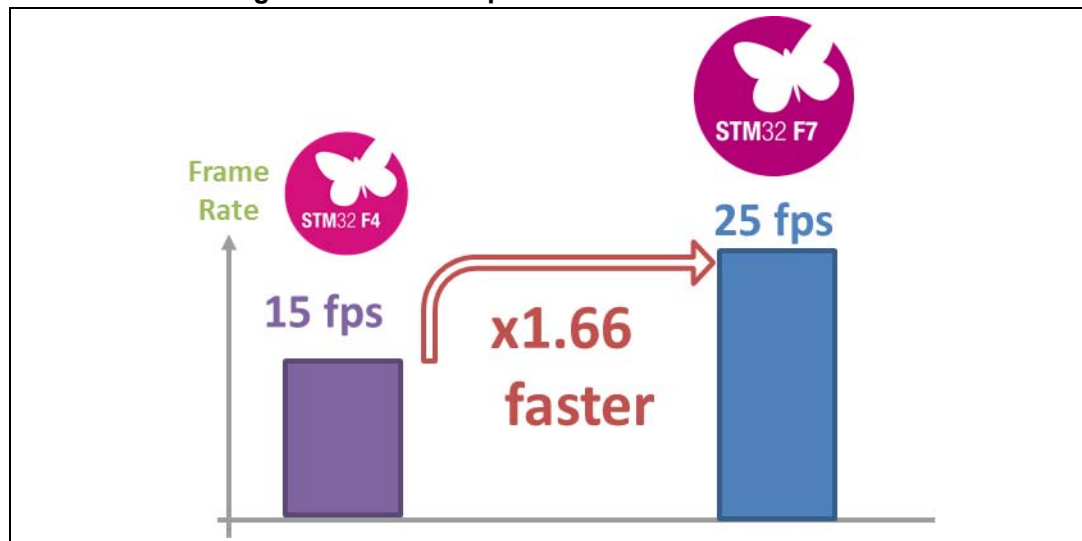
- 4 Kbytes dedicated for instruction cache
- 4 Kbytes dedicated for data cache

Figure 19. STM32F7 architecture



Using the STM32F7, the video performance is increased from 15fps to 25fps with QVGA resolution compared to STM32F4.

Figure 20. STM32F7 performance versus STM32F4



The instruction cache and data cache are enabled in the “main.c” file as shown in the code below:

```

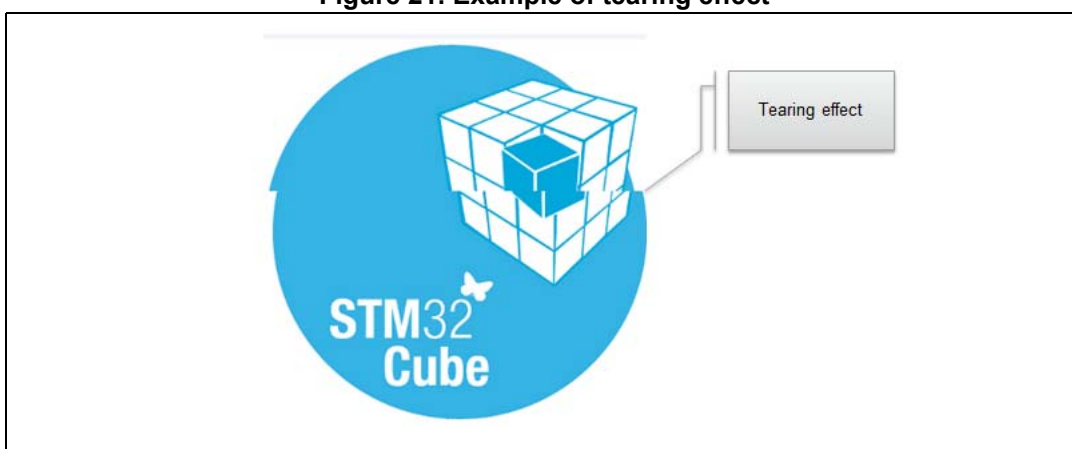
106  /**
107   * @brief Main program
108   * @param None
109   * @retval int
110   */
111  int main(void)
112  {
113      /* Configure the MPU attributes as Write Through */
114      MPU_Config();
115
116      /* Invalidate I-Cache : ICIALLU register*/
117      SCB_InvalidateICache();
118
119      /* Enable branch prediction */
120      SCB->CCR |= (1 <<18);
121      __DSB();
122
123      /* Invalidate I-Cache : ICIALLU register*/
124      SCB_InvalidateICache();
125
126      /* Enable I-Cache */
127      SCB_EnableICache();
128
129      SCB_InvalidateDCache();
130      SCB_EnableDCache();

```

6.2 Multi buffering features

Multiple buffering is the use of more than one frame buffer, so that the display ever shows a screen which is already completely rendered, even if a drawing operation is in process. When starting the process of drawing the current content of the front buffer is copied into a back buffer. After that all drawing operations take effect only on this back buffer. After the drawing operation has been completed the back buffer becomes the front buffer. Making the back buffer the visible front buffer normally only requires the modification of the frame buffer start address register of the display controller. Now it should be considered that a display is refreshed by the display controller approximately 60 times per second. After each period there is a vertical synchronization signal, known as VSYNC signal. The best moment to make the back buffer the new front buffer is this signal. If not considering the VSYNC signal tearing effects can occur, as shown in [Figure 21](#) below.

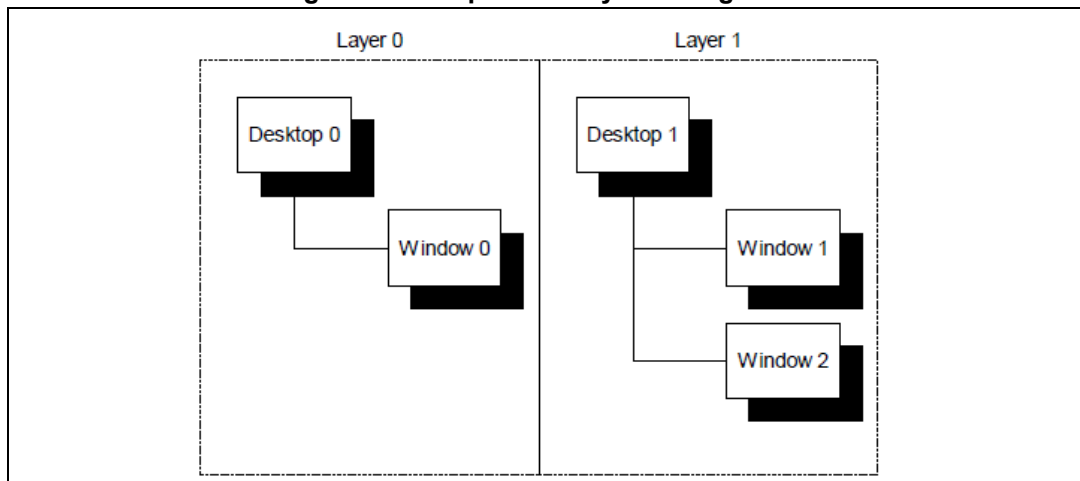
Figure 21. Example of tearing effect



6.3 Multi layers feature

Windows can be placed in any layer or display, drawing operations can be used on any layer or display. Since there are really only smaller differences from this point of view, multiple layers and multiple displays are handled the same way (using the same API routines) and are simply referred to as multiple layers, even if the particular embedded system uses multiple displays.

Figure 22. Independent layer management



6.4 Hardware acceleration

With the STM32F7 demonstrations, the hardware acceleration capabilities of the STM32F756/F746 devices are used. STemWin offers a set of customization callbacks to change the default behavior based on the hardware capabilities, the optimized processes are implemented in the LCDConf.c file and implement the following features:

a) Color conversion:

Internally STemWin works with logical colors (ABGR). To be able to translate these values into index values for the hardware and vice versa the color conversion routines automatically use the DMA2D for that operation if the layer work with direct color mode.

This low level implementation makes sure that in each case where multiple colors or index values need to be converted the DMA2D is used.

b) Drawing of index based bitmaps:

When drawing index based bitmaps STemWin first loads the palette of the bitmap into the DMA2Ds LUT instead of directly translating the palette into index values for the hardware. The drawing operation then is done by only one function call of the DMA2D.

c) Drawing of high color bitmaps:

If the layer works in the same mode as the high color bitmap has its pixel data available, these bitmaps can be drawn by one function call of the DMA2D. The following function is used to set up such a function:

```
LCD_SetDevFunc(LayerIndex, LCD_DEVFUNC_DRAWBMP_16BPP, pFunc);
```

d) Filling operations:

Setting up the function for filling operations:

```
LCD_SetDevFunc(LayerIndex, LCD_DEVFUNC_FILLRECT, pFunc);
```

e) Copy operations:

Setting up the functions for copy operations used by the function GUI_CopyRect():

```
LCD_SetDevFunc(LayerIndex, LCD_DEVFUNC_COPYRECT, pFunc);
```

f) Copy buffers:

Setting up the function for transferring the front- to the back buffer when using multiple buffers:

LCD_SetDevFunc(LayerIndex, LCD_DEVFUNC_COPYBUFFER, pFunc);

g) Fading operations:

Setting up the function for mixing up a background and a foreground buffer used for fading memory devices:

GUI_SetFuncMixColorsBulk(pFunc);

h) General alpha blending:

The following function replaces the function which is used internally for alpha blending operations during image drawing (PNG or true color bitmaps) or semitransparent memory devices:

GUI_SetFuncAlphaBlending(pFunc);

i) Drawing antialiased fonts:

Setting up the function for mixing single foreground and background colors used when drawing transparent ant aliased text:

GUI_SetFuncMixColors(pFunc).

7 Footprint

The purpose of the following sections is to provide the memory requirements for all the demonstration modules, including jpeg decoder and STemWin's main GUI components. The aim is to have an estimation of memory requirement in case of suppression or addition of a module or feature.

The footprint data are provided for the following environment:

- Tool chain: IAR 7.40.1
- Optimization: high size
- Boards: STM327x6G-EVAL, STM32746G-Discovery

[Table 7](#) shows the code memory, data memory and the constant memory used for each kernel file.

Table 7. Modules footprint

File	Code [byte]	Data [byte]	Const [byte]
Audio	9500	29012	422328 ⁽¹⁾
Audio recorder	2156	690	334336
VNC server	3452	431 + 15560 ⁽²⁾	457884
Video	5276	3717	1754819 ⁽³⁾
Home alarm	3340	56	5772536 ⁽³⁾
Garden control	1088	44	929905 ⁽³⁾
Games	3852	1936	245875 ⁽³⁾
System info	1252	340	1476719 ⁽³⁾

1. Some resources saved in NOR Flash memory for STM327x6G_EVAL demonstration. For STM32746G-Discovery all resources are stored in QSPI memory.
2. 15560 size of shared bytes.
3. All resources are stored in NOR Flash memory for STM327x6G-EVAL demonstration. For the STM32746G-Discovery, All resources are in QSPI.

7.1 STemWin features resources

7.1.1 JPEG decoder

The JPEG decompression uses approximately 33 Kbytes of RAM for the decompression independently of the image size and a size dependent amount of byte. The RAM requirement can be calculated as follows:

Approximate RAM requirement = X-Size of image * 80 bytes + 33 Kbytes

Table 8. RAM requirements for some JPEG resolutions

Resolution	RAM usage [kbyte]	RAM usage, size dependent [kbyte]
160x120	45.5	12.5
320x340	58.0	25.0

Table 8. RAM requirements for some JPEG resolutions (continued)

Resolution	RAM usage [kbyte]	RAM usage, size dependent [kbyte]
480x272	70.5	37.5
640x480	83.0	50.0

The memory required for the decompression is allocated dynamically by the STemWin memory management system. After drawing the JPEG image the complete RAM will be released.

7.1.2 GUI Components

The operation area of STemWin varies widely, depending primarily on the application and features used. In the following sections, memory requirements of various modules are listed, as well as the memory requirements of example applications.

[Table 9](#) shows the memory requirements of the main components of STemWin. These values depend a lot on the compiler options, the compiler version and the used CPU. Note that the listed values are the requirements of the basic functions of each module.

Table 9. MemoSTemWin components memory requirements

Component	ROM	RAM	Description
Windows Manager	6.2 Kbytes	2.5 Kbytes	Additional memory requirements of basic application when using the Windows Manager
Memory Devices	4.7 Kbytes	7 Kbytes	Additional memory requirements of basic application when using memory devices
Antialiasing	4.5 Kbytes	2 * LCD_XSIZE	Additional memory requirements for the antialiasing software item
Driver	2-8 Kbytes	20 bytes	The memory requirements of the driver depend on the configured driver and whether a data cache is used or not. With a data cache, the driver requires more RAM
Multilayer	2-8 Kbytes	-	If working with a multi layer or a multi display configuration, additional memory is required for each additional layer, because each requires its own driver
Core	5.2 Kbytes	80 bytes	Memory requirements of a typical application without using additional software items
JPEG	12 Kbytes	36 Kbytes	Basic routines for drawing JPEG files
GIF	3.3 Kbytes	17 Kbytes	Basic routines for drawing GIF files
Sprites	4.7 Kbytes	16 bytes	Routines for drawing sprites and cursors
Font	1-4 Kbytes	-	Depends on the font size to be used

Table 10. Widget memory requirements

Component	ROM	RAM	Description
BUTTON	1.0 Kbytes	40 bytes	(1)
CHECKBOX	1.0 Kbytes	52 bytes	(1)
DROPDOWN	1.8 Kbytes	52 bytes	(1)
EDIT	2.2 Kbytes	28 bytes	(1)
FRAMEWIN	2.2 Kbytes	12 bytes	(1)
GRAPH	2.9 Kbytes	48 bytes	(1)
GRAPH_DATA_XY	0.7 Kbyte	-	(1)
GRAPH_DATA_XY	0.6 Kbyte	-	(1)
HEADER	2.8 Kbytes	32 bytes	(1)
LISTBOX	3.7 Kbytes	56 bytes	(1)
LISTVIEW	3.6 Kbytes	44 bytes	(1)
MENU	5.7 Kbytes	52 bytes	(1)
MULTIEDIT	7.1 Kbytes	16 bytes	(1)
MULTIPAGE	3.9 Kbytes	32 bytes	(1)
PROGBAR	1.3 Kbytes	20 bytes	(1)
RADIOBUTTON	1.4 Kbytes	32 bytes	(1)
SCROLLBAR	2.0 Kbytes	14 bytes	(1)
SLIDER	1.3 Kbytes	16 bytes	(1)
TEXT	1.0 Kbytes	16 bytes	(1)
CALENDAR	0.6 Kbyte	32 bytes	(1)

1. The listed memory requirements of the widgets contain the basic routines required for creating and drawing the widget. Depending on the specific widget there are several additional functions available which are not listed in the table.

8 Demonstration functional description

8.1 Audio player

Overview

The audio player module provides a complete audio solution based on the STM32Fxxx and delivers a high-quality music experience. It supports playing music in WAV format but may be extended to support other compressed formats such as MP3 and WMA audio formats.

Features

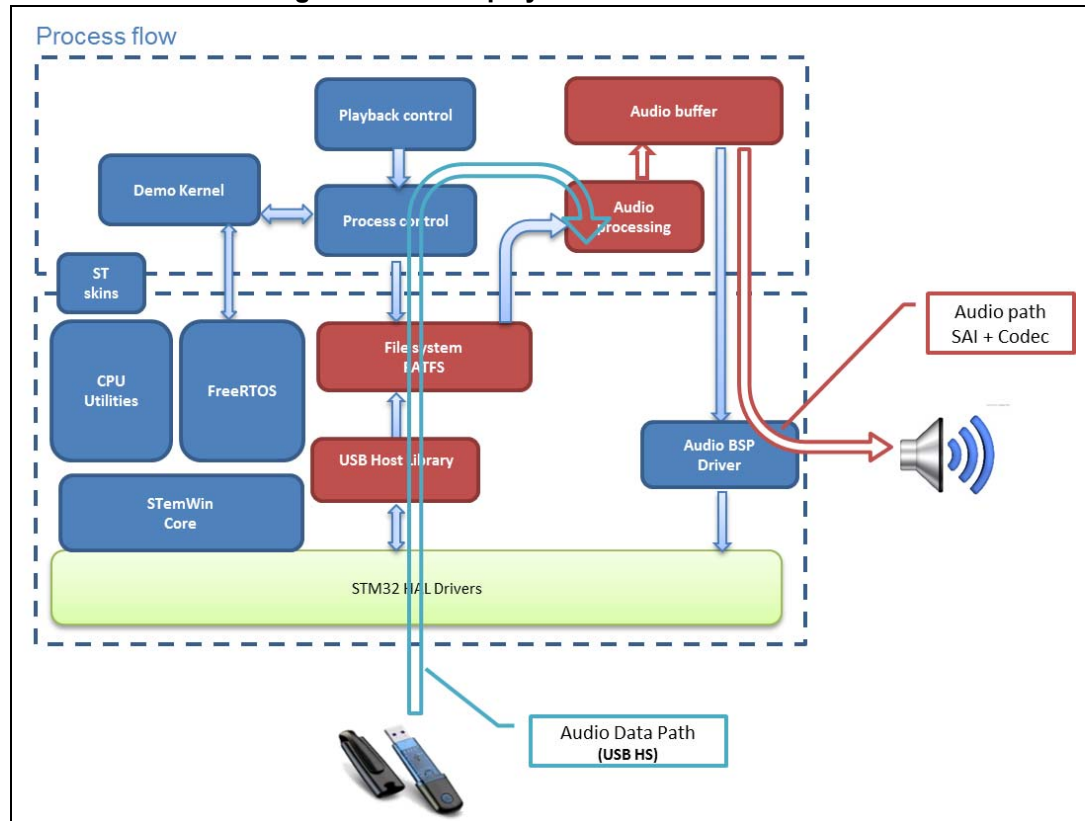
- Audio format: WAV format without compression with 8 k to 96 k sampling
- Embeds an equalizer and loudness control
- Performance: MCU load < 5%
- Audio files stored in USB Disk flash (USB High Speed)
- Supports background mode feature
- Only 8 Kbytes of RAM required for audio processing

MP3 format not supported but can be easily added (separate demo flavor).

Architecture

Figure 23 shows the different audio player parts and their connections and interactions with the external components.

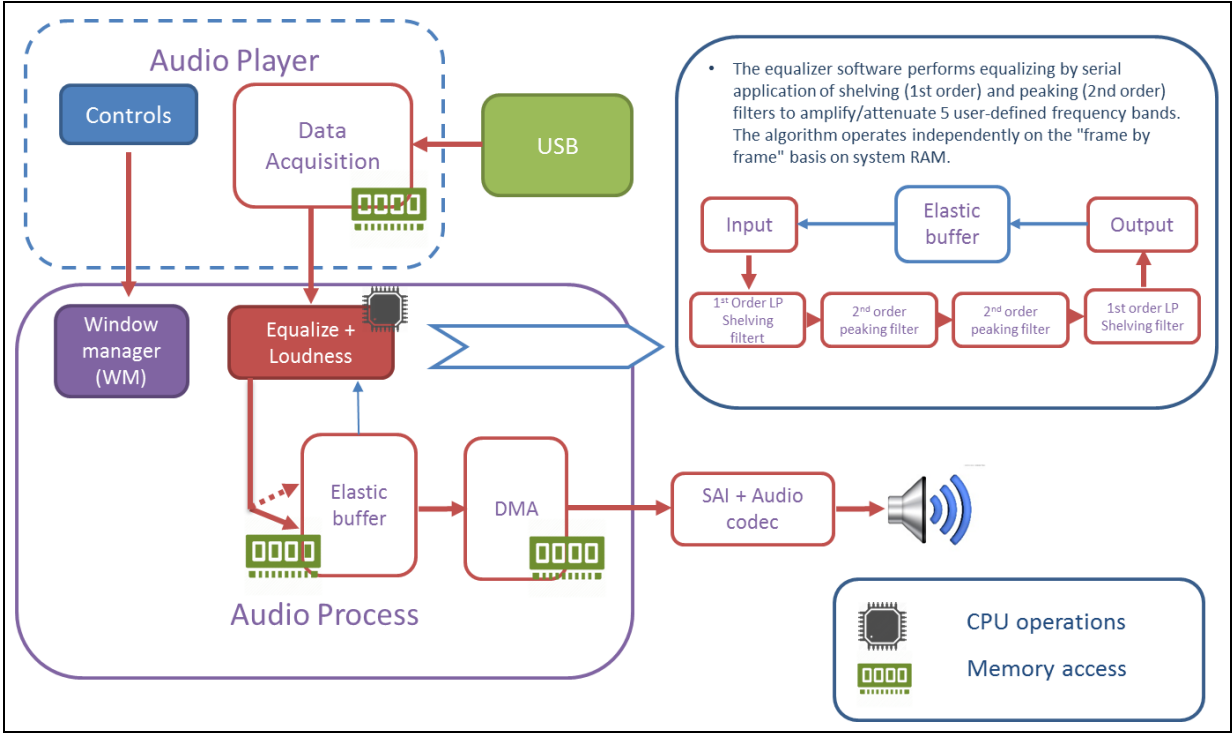
Figure 23. Audio player module architecture



Performance

Figure 24 shows the used performance mechanisms in audio process and audio player.

Figure 24. Audio player process

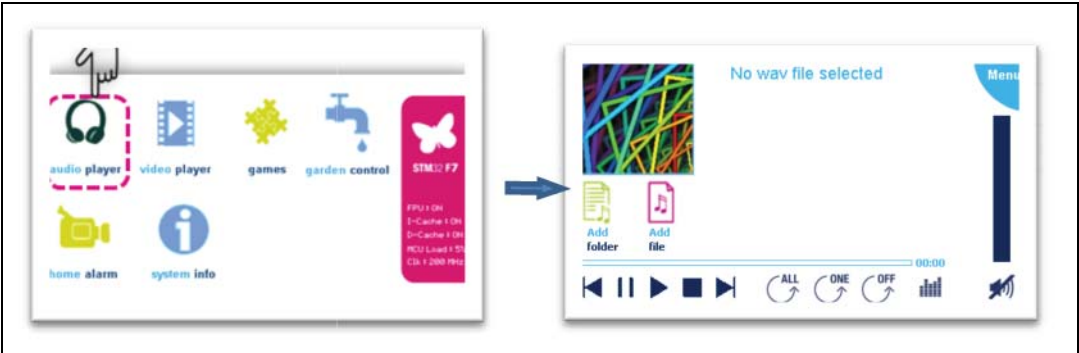


Process description

The audio player initialization is done in startup step. In this step all the audio player states, the speaker and the volume value are initialized and only when the play button in the audio player interface is pressed to start the process.

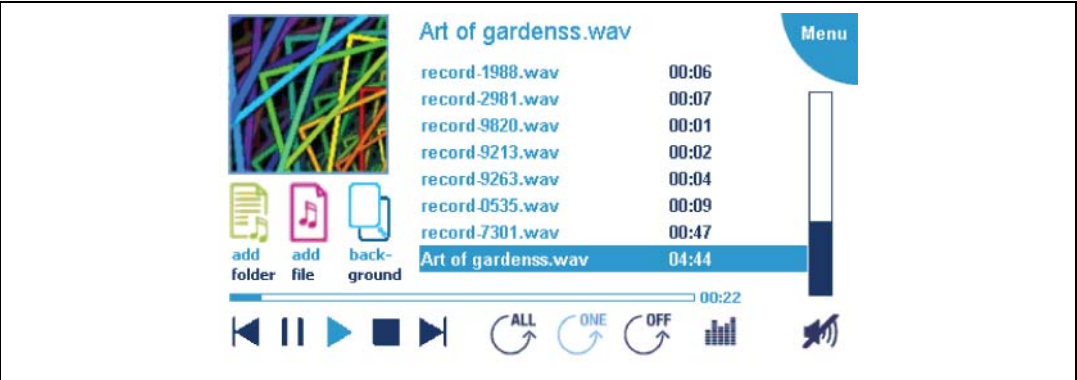
- Start audio player module from main desktop menu as shown in Figure 25 below:

Figure 25. Audio player module startup



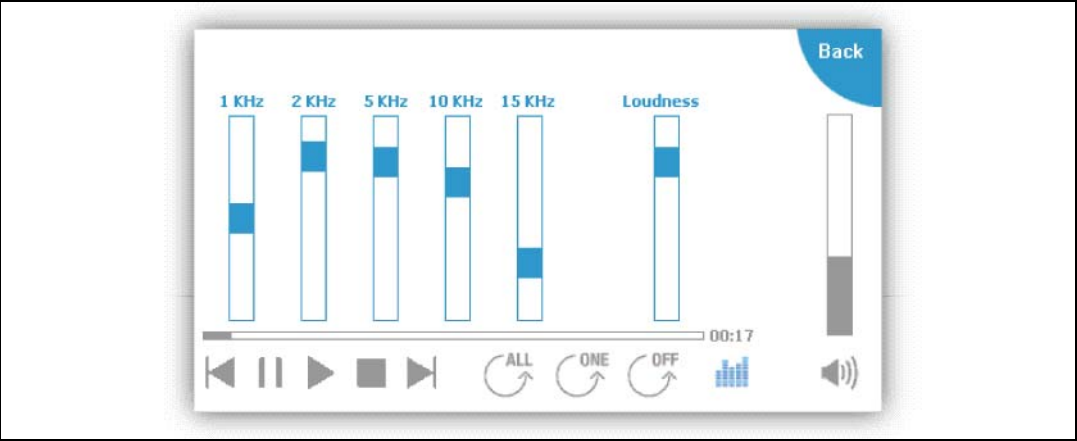
- Add audio file to play list

Figure 26. Audio player playlist



- Click on the Equalizer icon to open the equalizer and loudness frame

Figure 27. Equalizer frame



Hardware connectivity

Figure 28. Hardware connectivity



Table 11. Audio module controls

Button	Preview	Brief description
Play button		Changes the audio player state to "AUDIOPLAYER_PLAY". Reads the wave file from storage unit. Sets the frequency. Starts or resumes the audio task. Starts playing audio stream from a data buffer using "BSP_AUDIO_OUT_Play" function in BSP audio driver. Replaces play button by pause button.
Pause button		Suspends the audio task. Pauses the audio file stream. Replaces pause button by play button.
Stop button		Closes the wave file from storage unit. Suspends the audio task. Stops audio playing. Changes the audio player state to "AUDIOPLAYER_STOP".
Previous button		Points to the previous wave file. Stops audio playing. Starts playing the previous wave file if play button is pressed.
Next button		Points to the next wave file. Stops audio playing. Starts playing the next wave file if play button is pressed.
Add file to playlist button		Opens file browser window and chooses the wave file to be added to playlist.
Add folder button		Opens file browser window and chooses the entire folder to be added to playlist.

Table 11. Audio module controls (continued)

Button	Preview	Brief description
Repeat button		At the end of file: - If repeat all is selected the next wave file is selected and played. - If repeat once is selected the played wave file is repeated. - If repeat off is selected the audio player stop.
Speaker button		Sets the volume at mute (first press). Sets the volume at the value displayed in volume slider (second press).
Equalizer button		Starts the equalizer frame.
Menu button		Closes audio player module.

8.2 Audio recorder

Overview

The audio recorder module can be used to record audio frames in WAV format, save them in the storage unit and play them later.

Note: The Audio recorder module is only applicable to the STM32F746G-Discovery board.

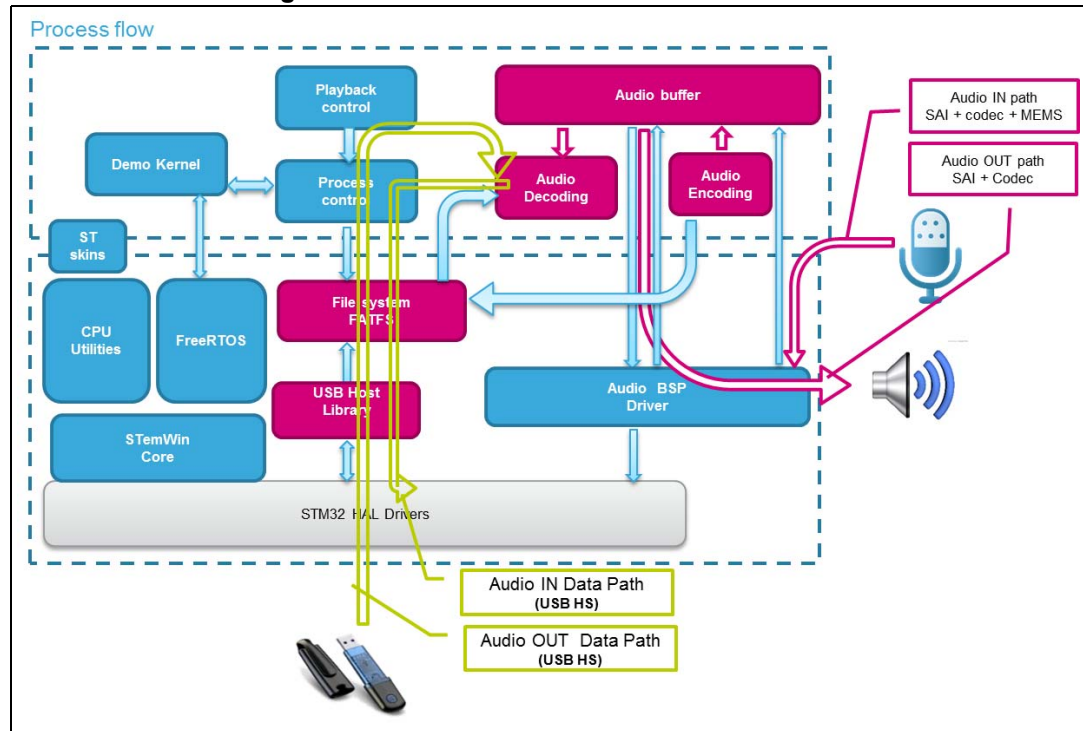
Features

- Audio Format: WAV format without compression with 16 k sampling stereo
- Performance: MCU Load < 5%
- Recorded files stored in USB Disk flash (USB High Speed)
- Embeds quick audio player
- Only 8 Kbytes of RAM required for audio processing
- **MP3 Format NOT supported but can be easily added (separate demo flavor)**

Architecture

Figure 29 shows the different audio recorder parts and their connections and interactions with the external components.

Figure 29. Audio recorder module architecture



Functional description

Start audio recorder module by touching the audio recorder icon. When the audio recorder is started, the following icon view is displayed.

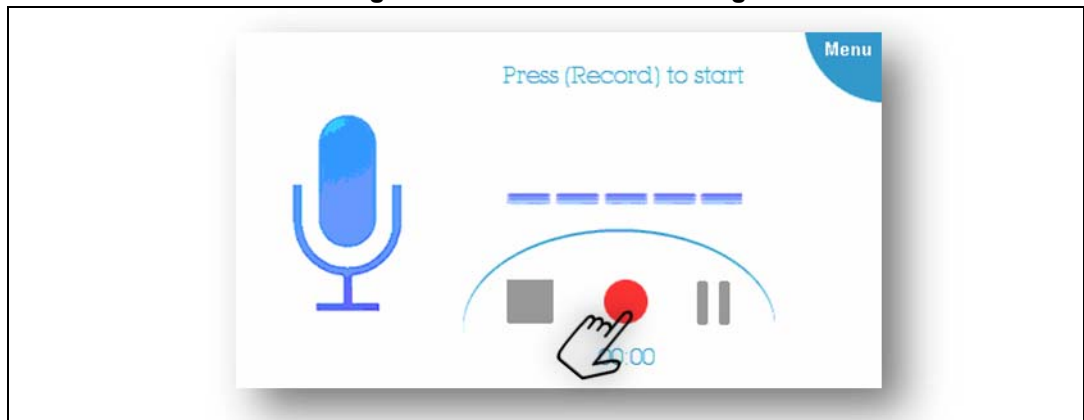
- Start audio recorder module from main desktop menu as shown in *Figure 30*:

Figure 30. Audio recorder module startup



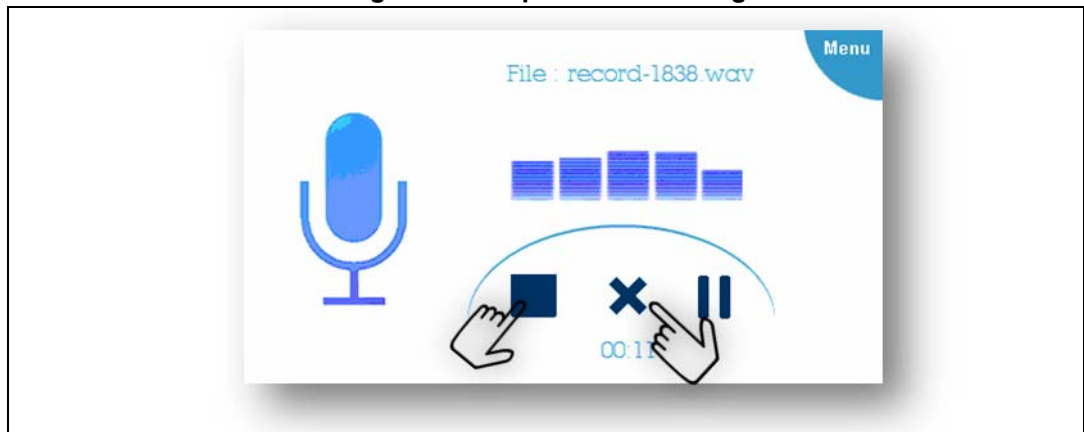
- Press on the record icon to start recording:

Figure 31. Start audio recording



- Click on Stop icon to save the recorded data in USB disk or click cancel to discard them:

Figure 32. Stop audio recording



- Click on Play to listen to the last recorded data or Click on stop to return to the recorder:

Figure 33. Play the recorded wave



Hardware connectivity

Figure 34. Hardware connectivity

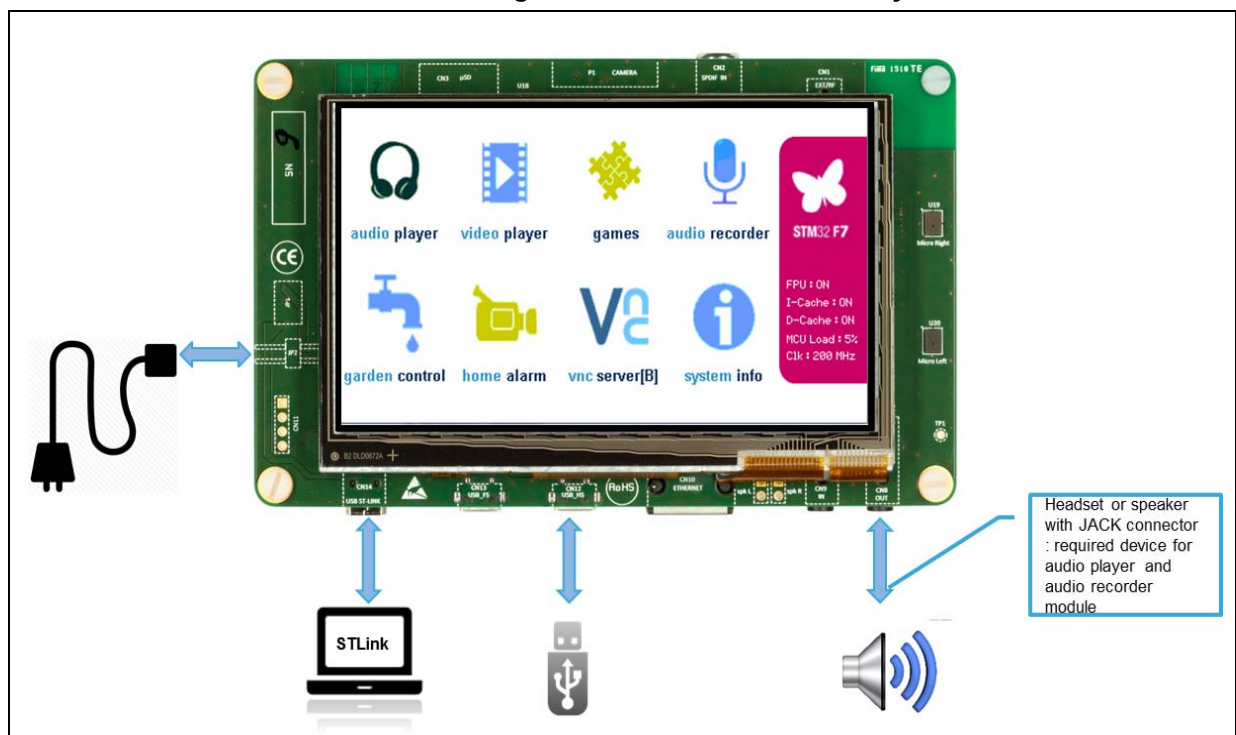


Table 12. Audio module controls

Button	Preview	Brief description
Play button		Reads the recorded wave file from the storage unit. Replaces Discard/start button by play button.
Pause button		Suspends the audio task. Pauses the audio file record.
Stop button		Saves the recorded file in the storage unit. Suspends the audio task. Stops audio recording.
Start button		Starts recording audio.
Cancel button		Stops audio recording. Discards the recorded wave.
Menu button		Closes audio recorder module.

8.3 VNC server

Overview

The VNC server module allows controlling the demonstration from a remote machine. It is based on the TCP/IP LwIP stacks. The background mode is supported.

Note: The VNC server module is only applicable to the STM32F746G-Discovery board.

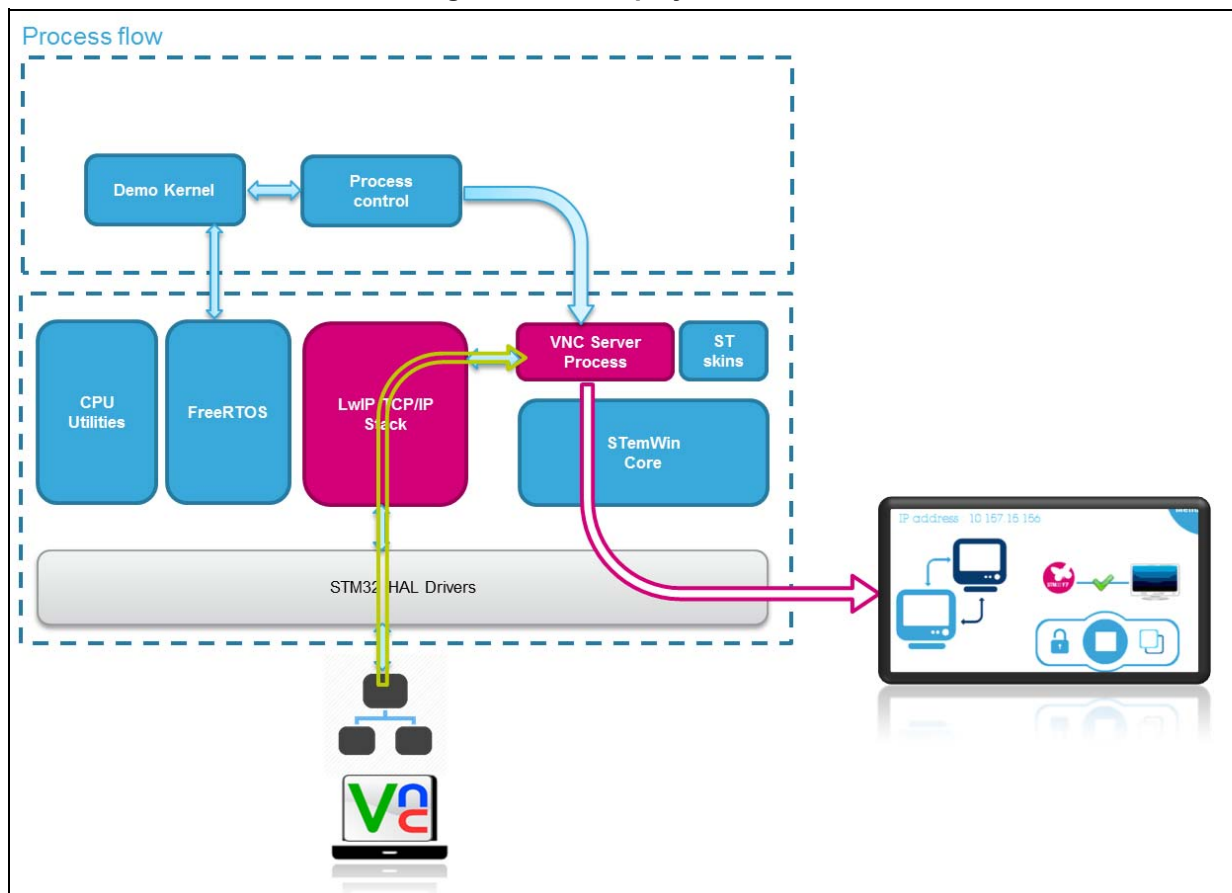
Features

- Based on the TCP/IP LwIP stacks (socket)
- IP address assigned by DHCP
- Secured mode supported (DES Encryption)
- Performance: MCU Load < 2 % (standalone)
- Background mode support
- Requires less than ~200 - 300 ms to update the entire display.

Architecture

Figure 35 shows the different VNC server module and their connections and interactions with the external components.

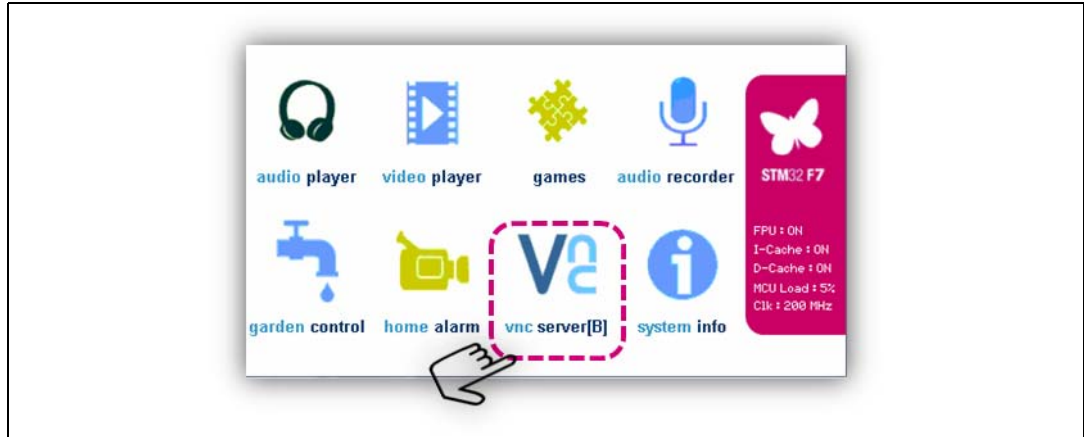
Figure 35. Video player module architecture



Functional description

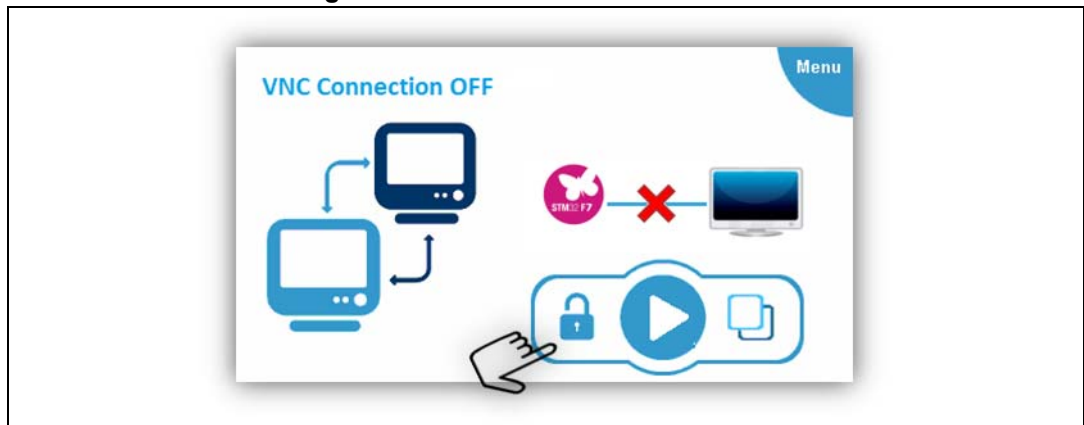
1. Click on the VNC server icon:

Figure 36. VNC server module startup



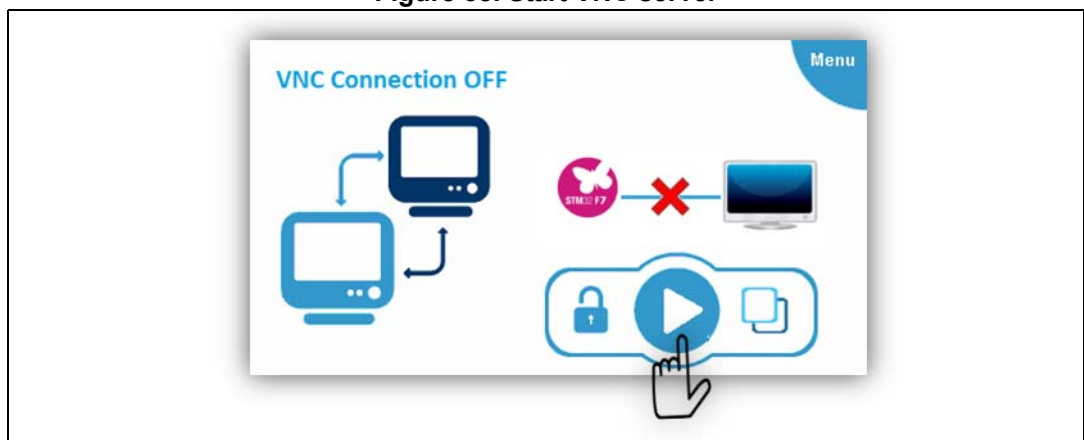
2. Enable or disable secure mode:

Figure 37. Enable/disable secure mode



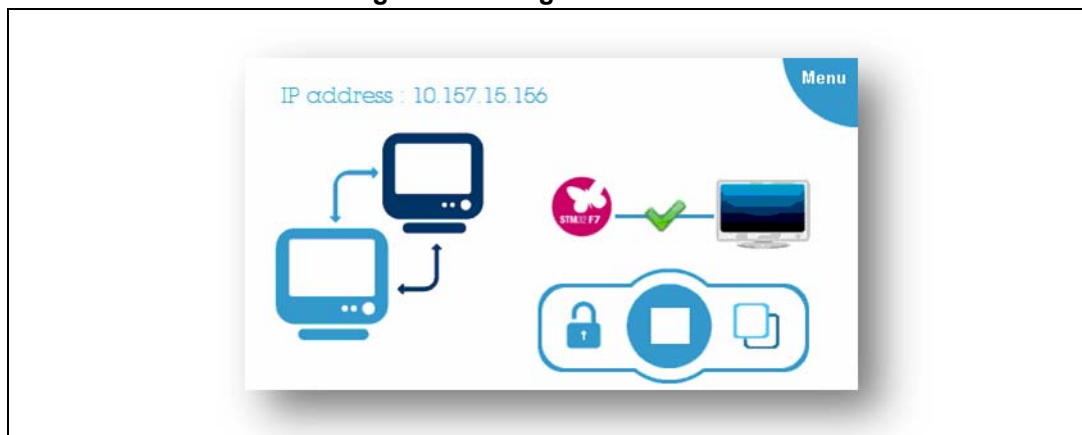
3. Start the VNC server:

Figure 38. Start VNC server



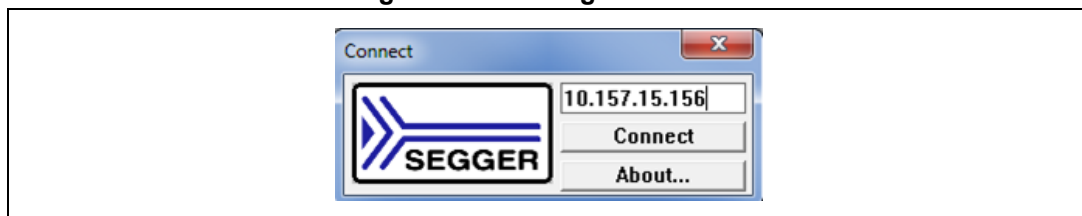
4. VNC connection established and IP address assigned:

Figure 39. Assigned IP address



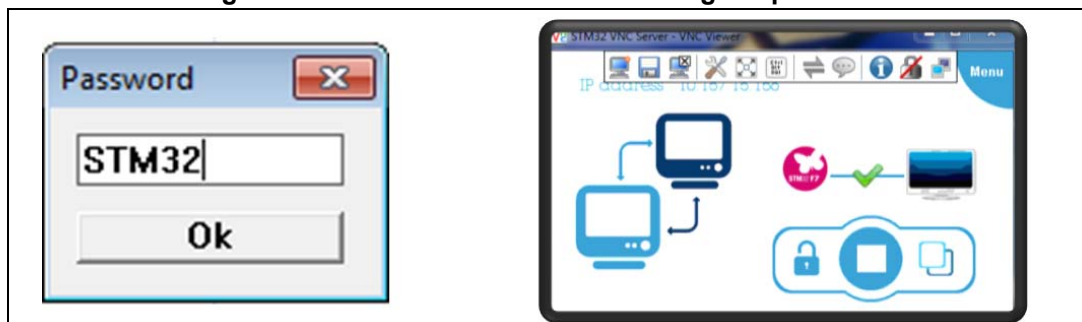
5. Run any VNC Client or the emVNC software, connect to server and enter the assigned IP address:

Figure 40. Entering IP address



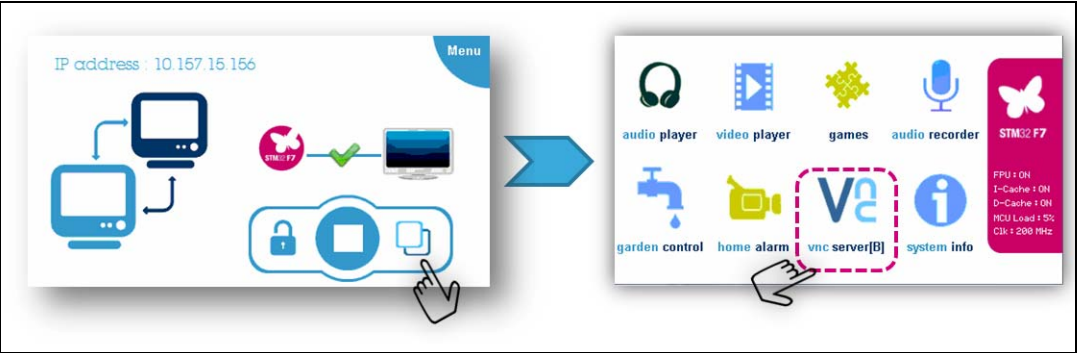
6. If secure mode is enabled a password is requested to establish a VNC connection. Enter the password "STM32" to display the demonstration content in VNC client:

Figure 41. Start VNC connection entering the password



7. Active background:

Figure 42. Background mode



Hardware connectivity

Figure 43. HW connectivity

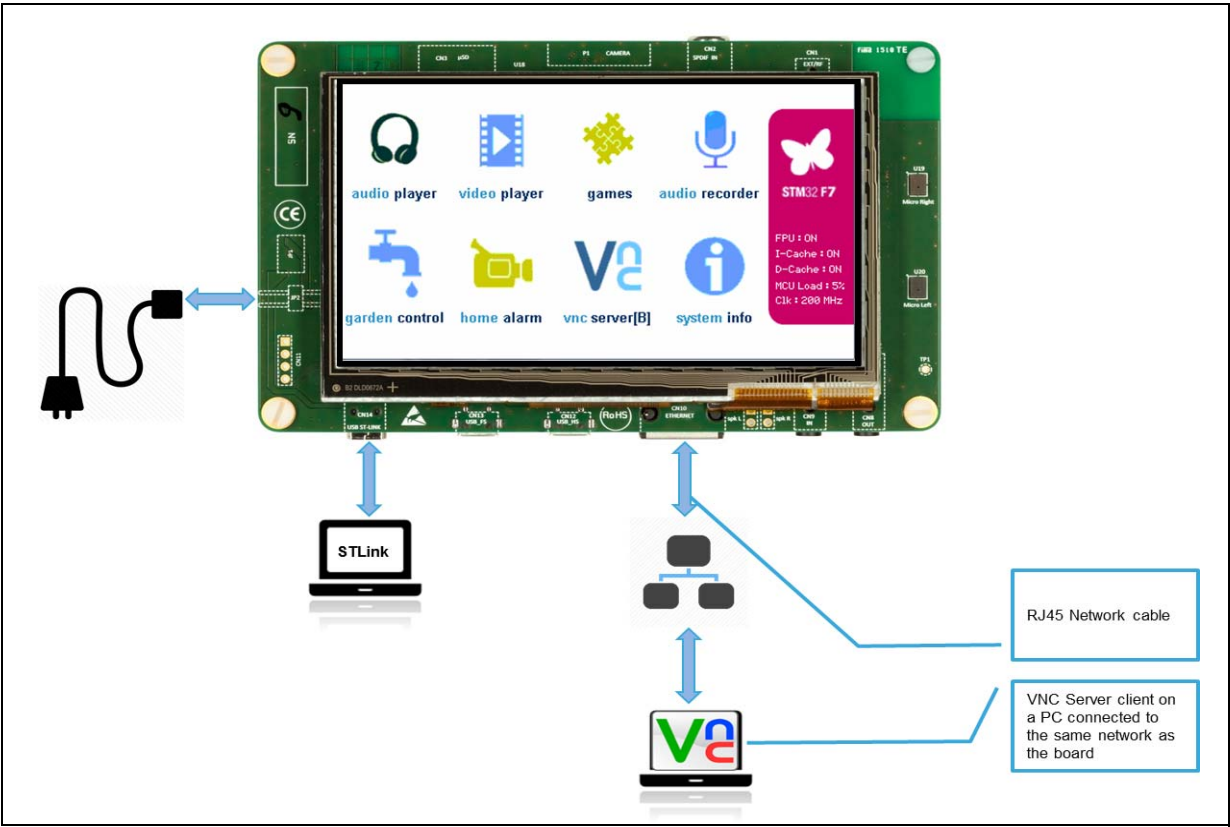


Table 13 summarizes the different actions behind each control button:

Table 13. Video module controls

Button	Preview	Brief description
Background button		Active background and any other module can be started
Stop button		Stops VNC server
Play button		Starts VNC server
Secure button		Enables or disables the secure mode
Menu button		Closes VNC server module

8.4 Video module

Overview

The video player module provides a video solution based on the STM32F7xxx and STemWin movie API. It supports playing movie in emf format.

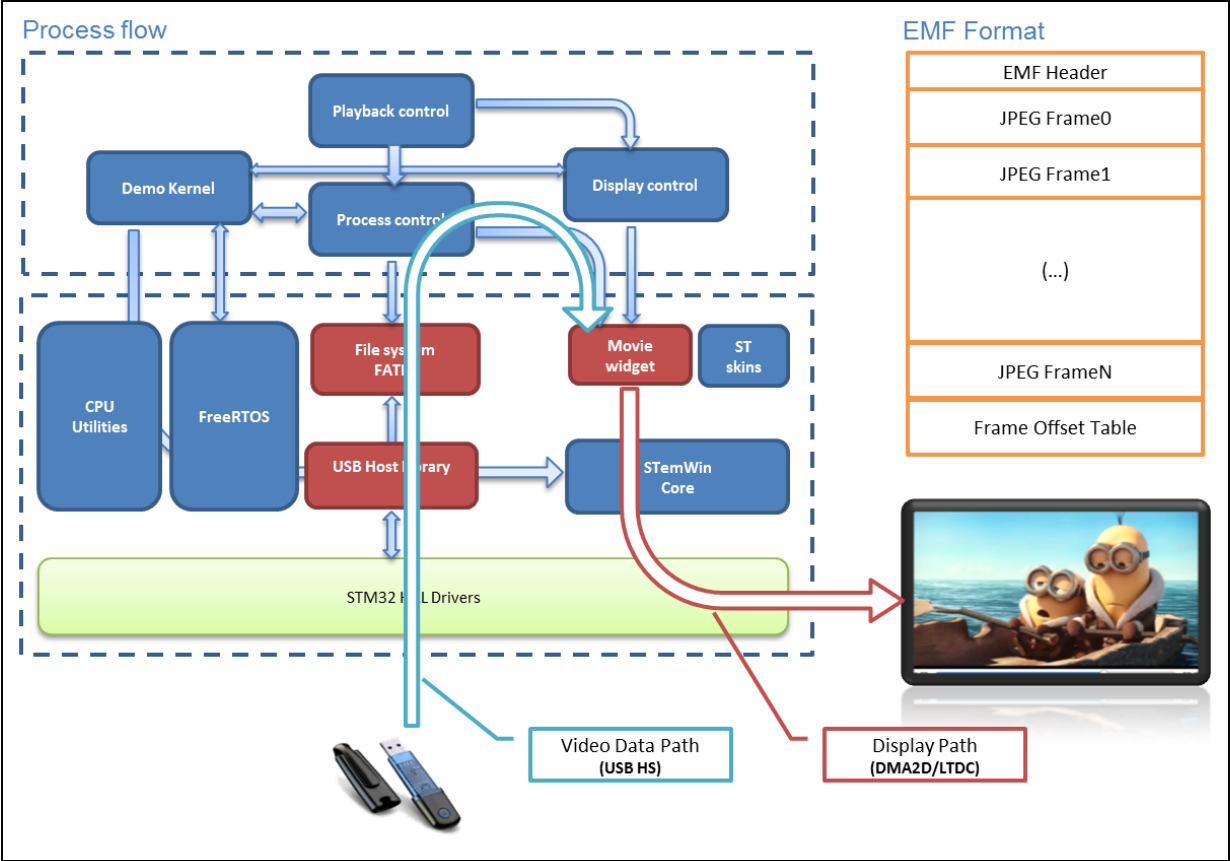
Features

- Video Format: STemWin emf Video Format (Motion-Jpeg)
- Performance: MCU Load < 70 % / Rate: up to 25 fps
- Video files stored in USB Disk Flash (USB High Speed)
- Use of the 2 LCD layers (playback control/ video display)
- 64 Kbytes of RAM required for JPEG decoding

Architecture

Figure 44 shows the different video player modules as well as their connections and interactions with the external components.

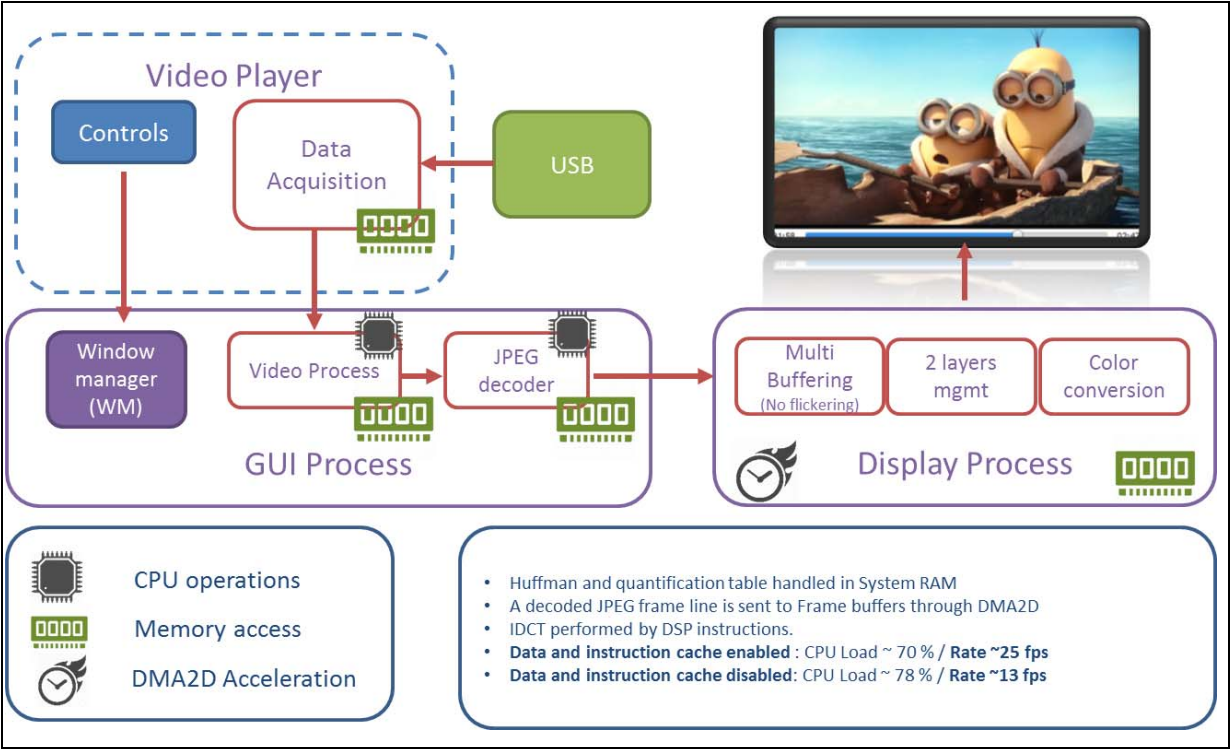
Figure 44. Video player module architecture



Performance

Figure 45 shows the GUI, display and video player process and performance.

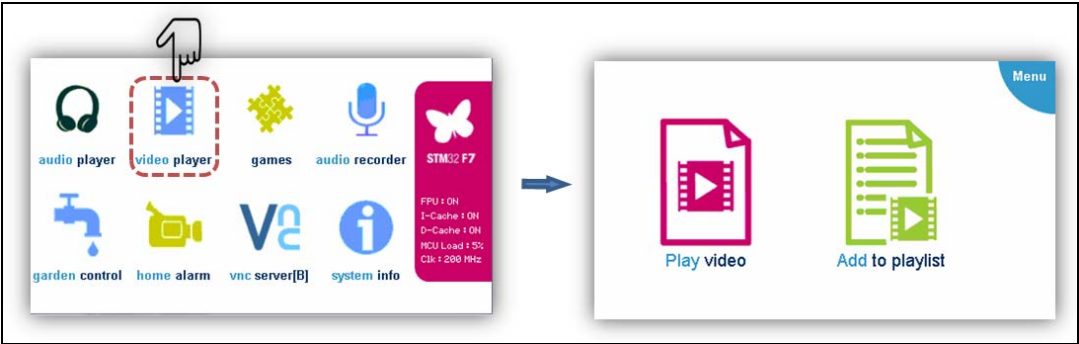
Figure 45. Video player process



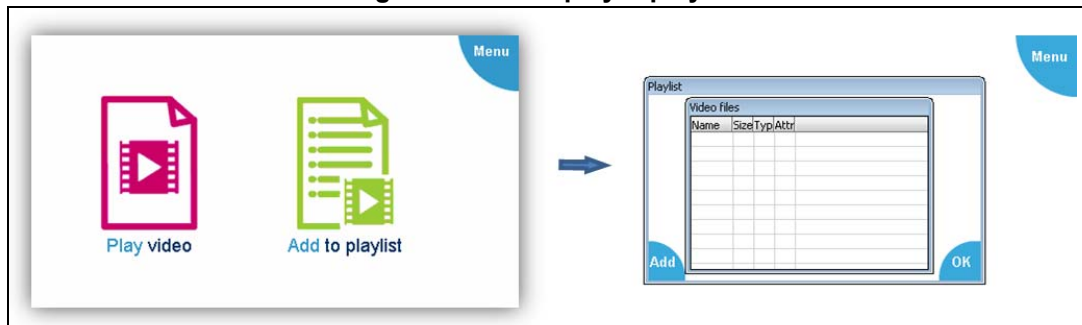
Functional description

1. Start video player module by touching the video player icon. When the video player is started, the following icon view is displayed.

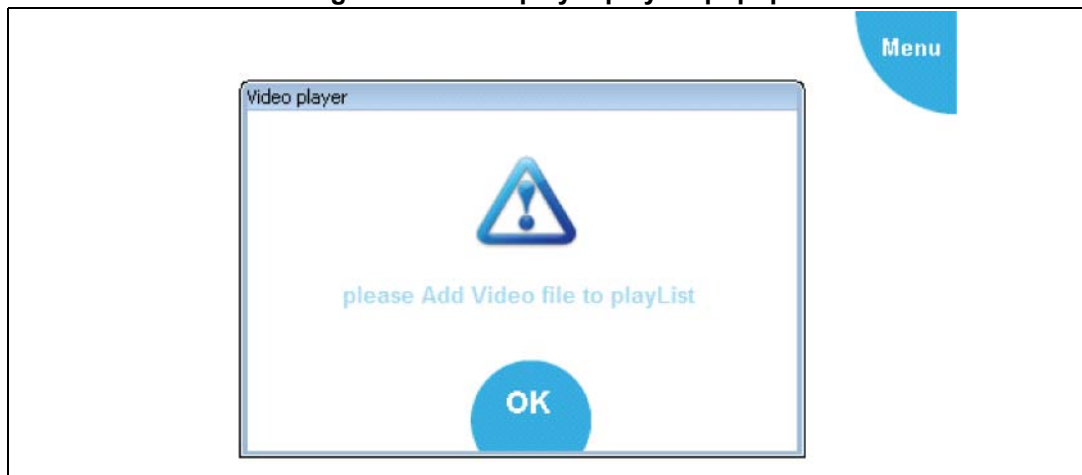
Figure 46. Video player module startup



2. Add video file to playlist by touching “Add to playlist” icon:

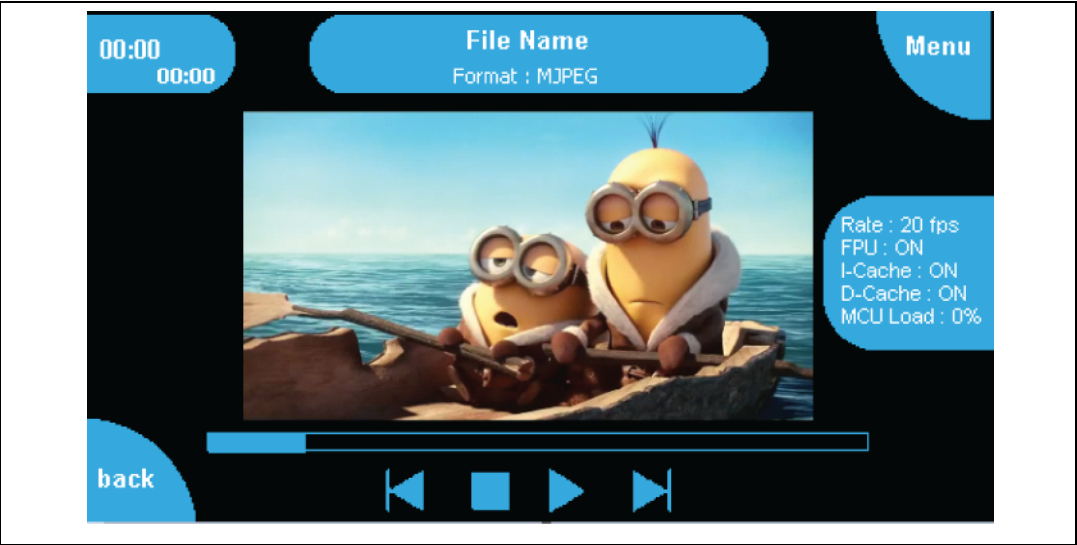
Figure 47. Video player playlist

3. Play video file by touching “Play video” icon. If there is no video file selected the following popup appears:

Figure 48. Video player playlist popup

Else, the video file starts playing:

Figure 49. Video player frame



- 4. Touch the screen to hide control keys, hardware information and video file information.

Figure 50. Video player control keys



Table 14 summarizes the different actions behind each control button:

Table 14. Video module controls

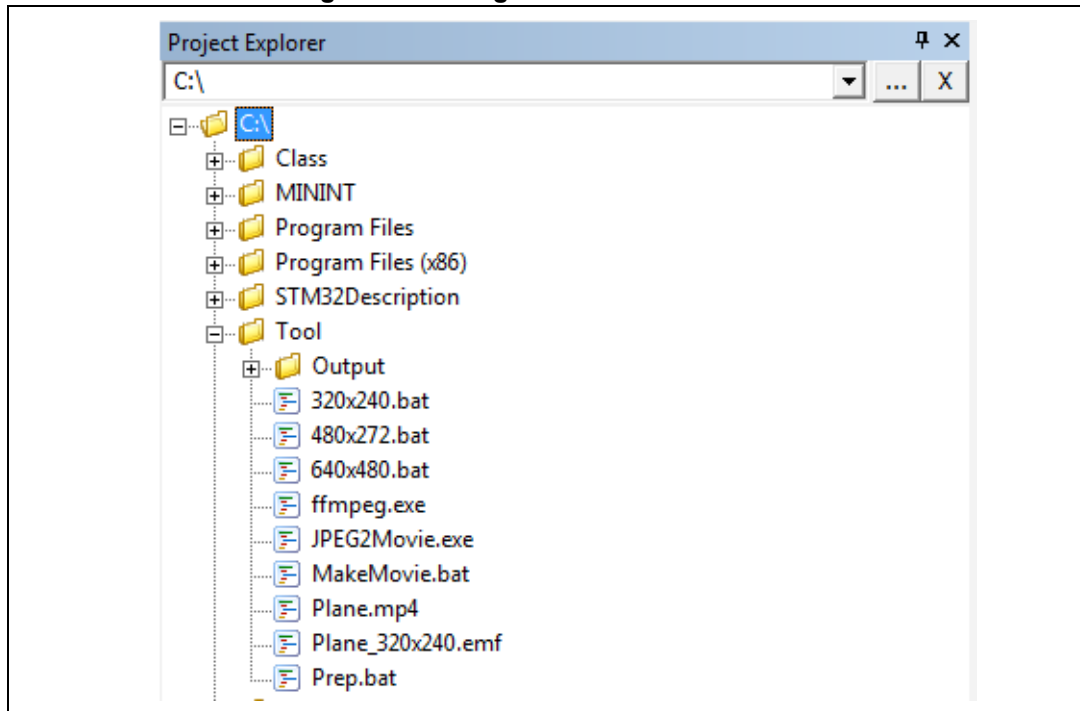
Button	Preview	Brief description
Play button		Checks if the video size is not supported. Supported video size: $0 < xSize < 1024$ and $0 < ySize < 768$. Changes the video player state to "VIDEO_PLAY". Reads the video file from storage unit. Replaces play button by pause button.
Pause button		Pauses the video file stream. Changes the video player state to "VIDEO_PAUSE". Replaces pause button by play button.
Next button		Points to the next video file. Stops video playing. Starts playing the next video file if play button is pressed.
Previous button		Points to the previous video file. Stops video playing. Changes the video player state to "VIDEO_IDLE".
Stop button		Closes the video file from storage unit. Stops video playing. Changes the video player state to "VIDEO_IDLE".
Back button		Back to previous video player frame to add new video file.
Menu button		Closes video player module.

Video file creation (emf)

To be able to play movies with the STemWin API functions it is required to create files of the STemWin specific EmWin movie file format. There are two steps to generate an emf file:

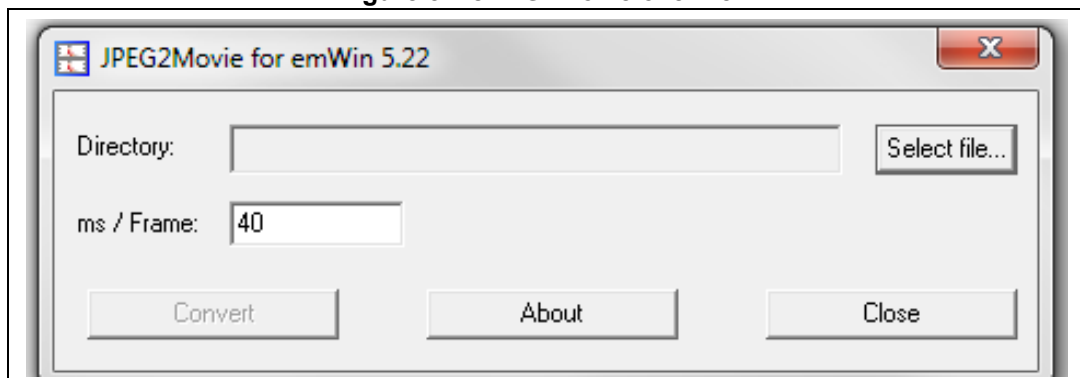
- a) Convert files of any MPEG file format into a folder of single JPEG files for each frame (*Figure 51*). The free FFmpeg available at [ffmpeg website](http://ffmpeg.org) can be used.

Figure 51. EMF generation environment



- b) Create an emf file from JPEG file using JPEG2Movie tool available in STemWin package (see [Figure 52](#)).

Figure 52. JPEG2Movie overview



The above steps can be run once using a predefined batch (included in the STemWin package) as shown in [Figure 53](#).

Figure 53. EMF file generation

```

Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

c:\Tool>480x272.bat Plane.mp4
ffmpeg version N-60797-g76dd01e Copyright (c) 2000-2014 the FFmpeg developers
  built on Feb 20 2014 22:06:56 with gcc 4.8.2 (GCC)
  configuration: --enable-gpl --enable-version3 --disable-w32threads --enable-av
  isynth --enable-bzlib --enable-fontconfig --enable-frei0r --enable-gnutls --enab
  le-iconv --enable-libass --enable-libbluray --enable-libcaca --enable-libfreetyp
  e --enable-libgsm --enable-libilbc --enable-libmodplug --enable-libmp3lame --ena
  ble-libopencore-amrnb --enable-libopencore-amrwb --enable-libopenjpeg --enable-l
  ibopus --enable-librtmp --enable-libschröedinger --enable-libsoxr --enable-libsp
  eex --enable-libtheora --enable-libtwolame --enable-libvidstab --enable-libvo-aa
  cenc --enable-libvo-amrwbenc --enable-libvorbis --enable-libvpx --enable-libwavp
  ack --enable-libx264 --enable-libx265 --enable-libxavs --enable-libxvid --enable
  -zlib
  libavutil      52. 65.100 / 52. 65.100
  libavcodec     55. 52.102 / 55. 52.102
  libavformat    55. 33.100 / 55. 33.100
  libavdevice    55. 10.100 / 55. 10.100
  libavfilter     4. 1.103 / 4. 1.103
  libswscale     2. 5.101 / 2. 5.101
  libswresample  0. 17.104 / 0. 17.104
  libpostproc   52. 3.100 / 52. 3.100
Input #0, mpeg, from 'Plane.mp4':
  Duration: 00:00:11.76, start: 0.480000, bitrate: 1397 kb/s
  Stream #0:0[0x1e01]: Video: mpeg1video, yuv420p(tv), 352x288 [SAR 49:33 DAR 4
  9:27], 1150 kb/s, 25 fps, 25 tbr, 90k tbn, 25 tbc
[suscaler @ 0000000002cd7fe0] deprecated pixel format used, make sure you did se
  t range correctly
Output #0, image2, to 'C:\Tool\Output\img-%05d.jpeg':
  Metadata:
    encoder      : Lavf55.33.100
  Stream #0:0: Video: mjpeg, yuvj420p, 480x272 [SAR 833:810 DAR 49:27], q=2-31
  , 200 kb/s, 90k tbn, 25 tbc
Stream mapping:
  Stream #0:0 -> #0:0 (mpeg1video -> mjpeg)
Press [q] to stop, [?] for help
frame= 295 fps=0.0 q=0.0 Lsize=N/A time=00:00:11.80 bitrate=N/A
video=2420kB audio=0kB subtitle=0 data=0 global headers=0kB muxing overhead -100
.000885%
C:\Tool\Output\Output.emf
  1 file(s) copied.
  
```

For more information about how to use the emf generation batches, refer to the STemWin user and reference guide (UM3001).

Table 15. Batch files description

File	Explanation
Prep.bat	Sets some defaults to be used. Needs to be adapted as explained in Prep.bat .
MakeMovie.bat	Main conversion file. Not to be adapted normally.
<X_SIZE>x<Y_SIZE>.bat	Some helper files for different resolutions. Detailed explanation in <X_SIZE>x<Y_SIZE>.bat

Prep.bat

The Prep.bat is required to prepare the environment for the actual process. Calling it directly will not have any effect. It is called by the MakeMovie.bat. To be able to use the batch files it

is required to adapt this file at first. This file sets variables used by the file MakeMovie.bat, they are listed in [Table 16](#).

Table 16. Variables description

Variable	Description
%OUTPUT%	Destination folder for the JPEG files. Will be cleared automatically when starting the conversion with MakeMovie.bat.
%FFMPEG%	Access variable for the FFmpeg tool. Should contain the complete path required to call FFmpeg.exe.
%JPEG2MOVIE%	Access variable for the JPEG2MOVIE tool. Should contain the complete path required to call JPEG2Movie.exe.
%DEFAULT_SIZE%	Default movie resolution to be used. Can be ignored if one of the <X-SIZE>x<Y-SIZE>.bat files are used.
%DEFAULT_QUALITY%	Default quality to be used by FFmpeg.exe for creating the JPEG files. The lower the number the better the quality. Value 1 indicates that a very good quality should be achieved, value 31 indicates the worst quality. For more details please refer to the FFmpeg documentation.
%DEFAULT_FRAMERATE%	Frame rate in frames/second to be used by FFmpeg. It defines the number of JPEG files to be generated by FFmpeg.exe for each second of the movie. For more details please refer to the FFmpeg documentation.

MakeMovie.bat

This is the main batch file used for the conversion process. Normally it is not required to change this file, but it is required to adapt Prep.bat first. It can be called with the parameters listed in [Table 17](#):

Table 17. Parameters description

Parameter	Description
%1	Movie file to be converted
%2 (optional)	Size to be used. If not given %DEFAULT_SIZE% of Prep.bat is used.
%3 (optional)	Quality to be used. If not given %DEFAULT_QUALITY% of Prep.bat is used.
%4 (optional)	Frame rate to be used. If not given %DEFAULT_FRAMERATE% of Prep.bat is used.

Since the FFmpeg output can differ strongly from the output of previous actions, the MakeMovie.bat deletes all output files in the first place. The output folder is defined by the environmental variable %OUTPUT% in Prep.bat. After that it uses FFmpeg.exe to create the required JPEG files for each frame. Afterwards it calls JPEG2Movie to create a single EMF file which can be used by STemWin directly. After the conversion operation the result

can be found in the conversion folder under FFmpeg.emf. It also creates a copy of that file into the source file folder. It will have the same name as the source file with a size-postfix and .emf extension.

<X_SIZE>x<Y_SIZE>.bat

These files are small but useful helpers if several movie resolutions are required. The filenames of the batch files itself are used as parameter '-s' for FFmpeg.exe. You can simply drag-and-drop the file to be converted to one of these helper files. After that an .emf file with the corresponding size-postfix can be found in the source file folder.

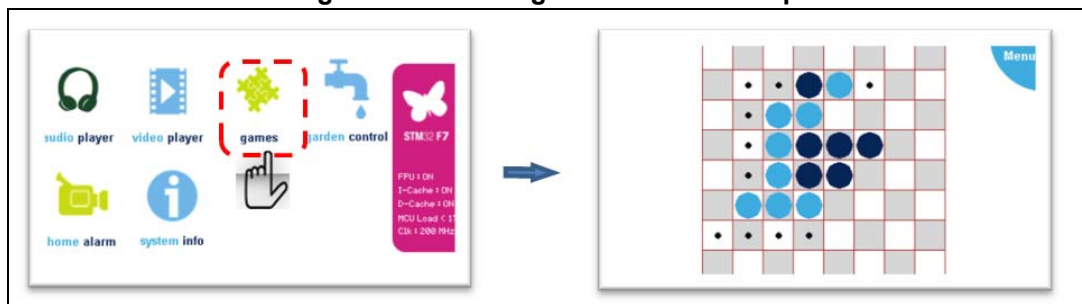
8.5 Game

overview

The game coming in the STM32CubeF7 demonstration is based on the Reversi game. It is a strategy board game for two players, played on an 8×8 board. The goal of the game is to have the majority of disks turned to display your color when the last playable empty square is filled.

In this STM32CubeF7 demonstration STM32 MCU is one of the two players. The GUI will ask the user to start a new game when the ongoing one is over.

Figure 54. Reversi game module startup

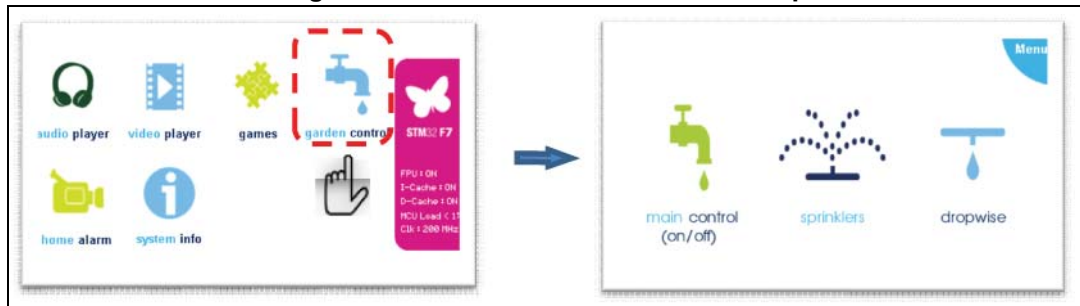


8.6 Garden control

overview

Control a garden watering system behavior, made with two independent circuits: one for a series of sprinklers and a drop wise system.

Figure 55. Garden control module startup



Note: This module is still in alpha version: only controls are shown in the main frame. Final version to come later.

8.7 Home alarm

overview

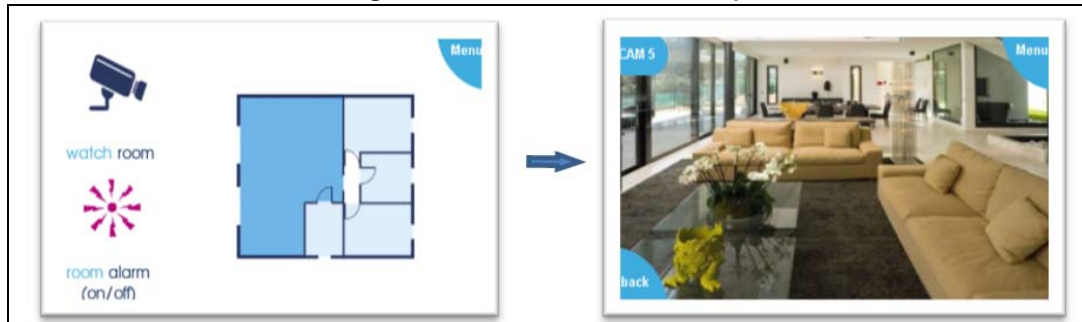
Home alarm system based on the integrated camera (in emulation mode fixed pictures are displayed for each room).

Figure 56. Home alarm module startup



Choose a room and click on “watch room” to show a static picture simulated as home camera.

Figure 57. Home camera startup



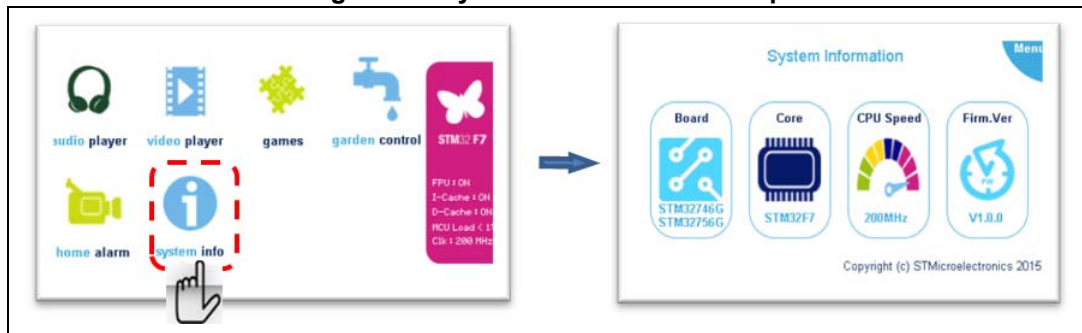
Note: Static pictures are used instead of camera streaming.

8.8 System Information

overview

The system information shows the main demonstration information such as: the used board, STM32F7 part number, and the current CPU clock and demonstration revision.

Figure 58. System information startup



9 Revision history

Table 18. Document revision history

Date	Revision	Changes
15-Jun-2015	1	Initial release.
07-Sep-2015	2	Added STM32746G-Discovery in the whole document: updating most of the paragraphs, updating the figure and Introduction in the cover, adding notes in Section 3.2: Kernel initialization and Section 3.4: Kernel graphical aspect. Updated Table 7: Modules footprint adding lines for audio recorder and VNC server, changing the notes. Updated Section 8: Demonstration functional description adding the audio recorder and VNC server modules. Updated Table 12: Audio module controls replacing 'Next button' by 'Cancel button'. Updated Section 3.13: Hardware settings changing Table 5: STM327x6G-EVAL board jumper configuration for demonstration title and adding a note.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2015 STMicroelectronics – All rights reserved